aws

開發人員指南

Amazon Rekognition

Amazon Rekognition: 開發人員指南

Table of Contents

什麼是 Amazon Rekognition

Amazon Rekognition 是雲端型影像和影片分析服務，可讓您輕鬆將進階電腦視覺功能新增至應用程式。此服務採用經過驗證的深度學習技術，不需要機器學習專業知識即可使用。Amazon Rekognition 包含一個簡單的 easy-to-use API，可以快速分析存放在 Amazon S3 中的任何影像或影片檔案。

您可以使用 Rekognition 的 API，新增偵測物件、文字、不安全內容、分析影像/視訊，以及比較臉孔與應用程式的功能。您可以偵測、分析及比對各種使用案例的臉部，包括驗證使用者、建立目錄、計算人數和公共安全。

這項服務採用 Amazon 電腦視覺科學家開發的經過驗證、可高度擴展的深度學習技術，該技術每天可分析數十億張影像和影片。Rekognition 會定期從新資料中學習，而且我們經常在服務中加入新的標籤和功能。

如需詳細資訊，請參閱 Amazon Rekognition 問答集。

關鍵能力

影像分析：

- 對象，場景和概念檢測-檢測和分類圖像中的對象，場景，概念和名人。
- 文本檢測-檢測和識別各種語言的圖像中的打印和手寫文本。
- 不安全內容-偵測並過濾露骨、不適當和暴力的內容和影像。偵測細微的不安全內容標籤。
- 名人識別-識別不同類別的圖像中成千上萬的名人，例如政客，運動員，演員和音樂家。
- 臉部分析-檢測，分析和比較臉部，以及面部屬性，例如性別，年齡和情緒。使用案例可能包括使用者驗證、編目、人流量統計和公共安全。
- 自訂標籤-建立自訂分類器，以偵測特定於您使用案例的物件，例如標誌、產品、字元。
- 圖像屬性-分析圖像屬性，如質量，顏色，清晰度，對比度。

影片分析：

- 對象，場景和概念檢測-檢測和分類視頻中的對象，場景，概念和名人。
- 文本檢測-檢測和識別各種語言的視頻中的打印和手寫文本。
- 人物路徑分析-追蹤已識別的人在視訊影格間移動時。

- 臉部分析-檢測，分析和比較流媒體或存儲視頻中的臉部。
- 名人識別-在不同類別（例如政客，運動員，演員和音樂家）中存儲的視頻中識別成千上萬的名人。
- 不安全內容偵測-偵測影片中的露骨、不適當和暴力內容。
- 視訊分割-自動識別有用的視訊片段，例如黑框和結束信用資訊。
- 面部活力-檢測面部驗證過程中是否存在實時用戶。

使用案例

可搜尋的媒體庫-Rekognition 可偵測影像和影片中的標籤、物件、概念和場景。您可以根據此視覺內容分析使這些標籤可搜尋。對於構建可搜索的圖像和視頻庫非常有用。

以臉部為基礎的使用者身分驗證-透過比較影像中的臉孔與參考臉部影像來確認使用者身 適用於應用程式中的身份驗證。

臉部活性偵測-Rekognition Face Liveness 是一項完全受管理的機器學習 (ML) 功能，旨在協助開發人員在臉部識別驗證期間阻止詐騙行為。該功能可幫助您驗證使用者是否實際存在於攝像頭前，並且不是利用使用者臉部詐騙的坏人。使用 Rekognition 臉部活體可協助您偵測對相機造成的詐騙攻擊，例如列印相片、數位相片/影片或 3D 遮罩。它還有助於檢測繞過攝像機的欺騙攻擊，例如直接注入視頻捕獲子系統的預錄或深度偽造視頻。

臉部搜尋-使用 Rekognition，您可以搜尋影像、儲存的影片和串流影片，找出符合儲存在臉孔集合容器中的臉孔。人臉集合為您持有與管理的臉部資料索引。根據人們的臉孔搜索需要您索引臉孔，然後搜索臉孔。

不安全內容偵測-偵測並過濾影像和影片中的露骨、不適當和暴力內容。根據業務需求使用標籤進行精細篩選。內容協調 API 也會傳回任何偵測到的標籤 (物件和概念) 的階層式清單，以及可信度分數。這些標籤可指出特定類別的不安全內容，並可對大量的使用者產生內容 (UGC) 執行精確篩選與管理。您可以使用配接器自訂內容協調 API 的輸出，如此可增強影像的效能，例如您提供作為訓練資料的影像。

偵測個人防護裝備-偵測圖像中的個人防護裝備，以監控各行各業的安全合規性。您可以透過偵測不當的設備來自動標記不安全的狀況，並接收有關這些狀況的警示，進而改善合規性和訓練。

名人識別-跨類別（例如政客，運動員，演員和音樂家）識別圖像和視頻中的名人。您可以識別名人外表，而無需提供名字。

文本檢測-檢測並提取圖像中的文本以進行可視化搜索或提取元數據。這適用於不同的字體和樣式。檢測方向以處理標誌和橫幅上的文本。

自訂標籤-識別特定於商業使用案例的自訂物件、概念和場景，例如標誌偵測。您可以訓練自訂分類器來處理利基或專有物件，這樣可以改善關鍵物件與一般分類器的準確度。如需詳細資訊，請參閱《Amazon Rekognition 自訂標籤開發人員指南》中的什麼是 Amazon Rekognition 自訂標籤？。

優勢

將強大的圖像和視頻分析集成到您的應用程序中-在沒有專業知識的情況下為應用添加準確的圖像 Amazon Rekognition API 可透過深度學習進行分析，而不需要任何機器學習知識。您可以在網頁、行動裝置和裝置應用程式中快速建置電腦視覺。

基於深度學習的圖像和視頻分析-使用深度學習分析圖像和視頻以實現高準確性。Amazon Rekognition 可以檢測標籤，物體，場景，臉部，名人。篩選結果以包含/排除特定標籤。

可擴展的圖像分析-分析數百萬張圖像以組織大量的可視化數據集。擴展以處理不斷增長的圖像庫和流量。您不需要規劃容量，而且只需按使用量付費。

根據屬性分析和過濾圖像-按屬性（例如質量，顏色和視覺內容）分析和過濾圖像，並檢測圖像的清晰度，亮度和對比度。

與其他 AWS 服務整合-Amazon Rekognition 整合現成的 S3 和 Lambda。您可以從 Lambda 呼叫 Amazon Rekognition 的 API，並在 Amazon S3 中處理映像，而無需移動資料。Rekognition 具有使用 IAM 的內建可擴展性和安全性。 AWS

低成本-P ay-as-you-go 定價，沒有最低限度或承諾。免費方案可供開始使用。透過分層定價節省更多用量表。相對於內部解決方案的成本效益。

簡單的定制-使用適配器為您的使用案例定制準確性。提供範例影像以訓練介面卡。改進了對給定域的對象和標籤檢測。無需 ML 專業知識即可輕鬆量身打造分析。

如需詳細資訊，請參閱 Amazon Rekognition 問答集。

Amazon Rekognition 和 HIPAA 資格

此為 HIPAA 合格服務。如需有關 AWS《1996 年美國 Health 保險流通與責任法案》(HIPAA)，以及使用 AWS 服務來處理、儲存和傳輸受保護的健康資訊 (PHI) 的詳細資訊，請參閱 HIP AA 概觀。

第一次使用 Amazon Rekognition 嗎？

若是第一次使用 Amazon Rekognition，建議您依序閱讀以下章節：

1. Amazon Rekognition 的運作方式— 本節介紹您用來建立體驗的各種 Amazon Rekognition 元件。end-to-end
2. Amazon Rekognition 入門：在本節中，您會設定帳戶、安裝反映您選擇語言的開發套件，然後測試 Amazon Rekognition API。如需 Amazon Rekognition 支援的程式設計語言清單，請參閱 搭配 SDK 使用 Rekognition AWS。
3. 使用映像：本節提供使用 Amazon Rekognition 分析儲存於 Amazon S3 儲存貯體中的映像以及從本機檔案系統中載入的映像等相關資訊。
4. 使用儲存的影片分析：本節提供使用 Amazon Rekognition 分析儲存於 Amazon S3 儲存貯體中的影片之相關資訊。
5. 使用串流視訊事件：本節提供使用 Amazon Rekognition 分析串流影片之相關資訊。

Amazon Rekognition 的運作方式

Amazon Rekognition 提供兩個 API 集進行視覺化分析：

- 用於圖像分析的 Amazon Rekognition 圖像
- Amazon Rekognition Video 視頻用於視頻分析

影像分析

使用 Amazon Rekognition 映像，您的應用程式可以：

- 偵測影像中的物件、場景和概念
- 認識名人
- 偵測各種語言的文字
- 偵測露骨、不適當或暴力的內容或影像
- 偵測、分析和比較臉部和面部屬性，例如年齡和情緒
- 偵測個人防護裝備的存在

使用案例包括增強相片應用程式、編目影像和協調內容。

影片分析

有了 Amazon Rekognition Video，您的應用程式可以：

- 跨視訊影格追蹤人員和物件
- 辨識物件
- 認識名人
- 搜索存儲和流式傳輸視頻感興趣的人
- 分析臉孔的屬性，如年齡和情緒
- 偵測露骨、不適當或暴力的內容或影像
- 依時間戳記和區段彙總和排序分析結果
- 偵測串流影片中的人物、寵物和包裹

使用案例包括影片分析、編目影片，以及篩選不適當的內容。

主要功能

- 強大的深度學習分析
- 對物體、場景、臉部、文字進行高精度偵測
- 易於使用的 API 集成到應用程序
- 根據您的資料調整的自訂模型
- 媒體庫的可擴展分析

Amazon Rekognition 可讓您訓練自訂介面卡，以提升特定深度學習模型的準確性。例如，使用 Amazon Rekognition Custom Moderation，您可以使用您的映像訓練自訂轉接器來調整 Amazon Rekognition 的基本映像分析模型。如需詳細資訊，請參閱使用自訂協調提高準確性。

以下各節涵蓋 Amazon Rekognition 提供的分析類型，以及 Amazon Rekognition 知影像和亞 Amazon Rekognition Video 操作的概觀。同時也將說明非儲存體與儲存體操作間的差別。

若要示範 Amazon Rekognition API，您可以看到步驟 3：開始使用 AWS CLI 和 AWS 開發套件 API，其中包括在主控台中試用 Rekognition。 AWS

主題

分析類型

以下是 Amazon Rekognition Image API 和 Amazon Rekognition Video API 可以執行的分析類型。如需 API 的相關資訊，請參閱 映像和影片操作。

下表列出您需要針對您正在使用的媒體類型和使用案例進行的操作：

使用案例	媒體類型	操作
管制內容	映像	DetectModerationLabels, StartMediaAnalysisJob,

使用案例	媒體類型	操作
		GetMediaAnalysisJob, ListMediaAnalysisJobs
	已存放影片	StartContentModeration, GetContentModeration
身分驗證	映像	CreateCollection, CreateUser, IndexFaces, AssociateFaces, SearchFacesByImage, SearchUsersByImage
	已存放影片	CreateCollection, IndexFaces, StartFaceSearch, GetFaceSearch
	串流視訊 (偵測人臉活體)	CreateFaceLivenessSession, StartFaceLivenessSession, GetFaceLivenessSessionResults,
臉部分析	映像	DetectFaces, CompareFaces
	已存放影片	StartFaceDetection, GetFaceDetection
	串流視訊	CreateStreamProcessor, StartStreamProcessor
物件和活動識別	映像	DetectLabels
	已存放影片	StartLabelDetection, GetLabelDetection
連接的家庭	串流視訊	StartStreamProcessor
媒體分析	已存放影片	StartSegmentDetection, GetSegmentDetection

使用案例	媒體類型	操作
職場安全	映像	DetectProtectiveEquipment
文字偵測	映像	DetectText
	影片	StartTextDetection, GetTextDetection
人物路徑	影片	StartPersonTracking, GetPersonTracking
名人辨識	映像	RecognizeCelebrities
	影片	StartCelebrityRecognition, GetCelebrityRecognition
自訂標籤偵測	映像	DetectCustomLabels
	模型訓練	請參閱自訂標籤開發人員指南

標籤

標籤是指下列任何項目：物件 (例如，花朵、樹或桌子)、事件 (例如，結婚、畢業典禮或生日派對)、概念 (例如，景色、黃昏和自然) 或活動 (例如，跑步或打籃球)。Amazon Rekognition 可以偵測映像與視訊中的標籤。如需詳細資訊，請參閱 偵測物件和概念。

Rekognition 可以檢測圖像和存儲的視頻中的大量標籤列表。Rekognition 還可以在串流視訊中偵測少量標籤。

根據您的使用案例，使用以下操作偵測標籤：

- 要檢測圖像中的標籤：使用DetectLabels。您可以識別映像屬性，例如主要映像顏色和映像品質。要實現這一目標，請DetectLabels與IMAGE_PROPERTIES作為輸入參數一起使用。
- 要檢測存儲視頻中的標籤：使用StartLabelDetection。儲存的視訊不支援偵測主要映像色彩和映像品質。
- 若要偵測串流視訊中的標籤：使用CreateStreamProcessor。串流視訊不支援偵測主要映像色彩和映像品質。

您可以使用包含性和排斥性篩選的選項，指定要針對映像和儲存的視訊標籤偵測傳回哪些類型的標籤。

自訂標籤

Amazon Rekognition 自訂標籤可以藉由培訓機器學習模型，識別出映像中專屬於您業務需求的物件和場景。例如，您可以培訓模型以偵測標誌或偵測裝配線上的工程機具零件。

> **Note**
>
> 有關 Amazon Rekognition 自訂標籤的資訊，請參閱《Amazon Rekognition 自訂標籤開發人員指南》https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/what-is.html。

Amazon Rekognition 提供您用來建立、培訓、評估和執行機器學習模型的主控台。如需詳細資訊，請參閱 Amazon Rekognition 自訂標籤開發指南中的 Amazon Rekognition 自訂標籤入門。您也可以使用 Amazon Rekognition 自訂標籤 API 來培訓和執行模型。如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的 Amazon Rekognition 自訂標籤開發套件入門。 CustomLabels

若要使用訓練過的模型分析影像，請使用DetectCustomLabels。

臉部活體檢測

Amazon Rekognition 臉部活體可協助您確認正在進行臉部型身分驗證的使用者實際存在於攝影機前方，而且不是詐騙使用者臉孔的壞人。其可以檢測出現給攝像機的欺騙攻擊以及繞過攝像機的攻擊。用戶可以通過拍攝短視頻自拍來完成臉部活體檢查，並返回活體分數以進行檢查。臉部活體是透過機率計算來決定的，且在檢查後傳回可信度分數 (介於 0 至 100 之間)。得分越高，接受检查的人的可信度就越大。

如需有關臉部活體的詳細資訊，請參閱 偵測人臉活體。

臉部偵測與分析

Amazon Rekognition 可以偵測映像與儲存的影片中的臉部。透過 Amazon Rekognition，您可以取得下列相關資訊：

- 在映像或視訊中偵測到臉孔的位置
- 臉部特徵點，例如左眼的位置。
- 圖像中存在臉部遮擋

- 偵測到情緒，例如快樂或悲傷
- 一個人在圖像中凝視的眼睛方向

您還可以解釋和人口統計信息，例如性別或年齡。您也可以將映像中的臉部與另一張映像中偵測到的臉孔進行比對。關於臉部的資訊也可儲存提供日後使用。如需詳細資訊，請參閱 偵測與分析人臉。

若要偵測影像中的臉部，請使用 DetectFaces。若要偵測儲存的影片中的臉部，請使用 StartFaceDetection。

臉部搜尋

Amazon Rekognition 可以搜尋臉孔。臉部資訊以索引編入容器中，也就是集合。集合中的臉部資訊可與映像、儲存的影片以及串流視訊中偵測到的臉孔進行比對。如需詳細資訊，請參閱 在集合中搜尋人臉。

要搜尋影像中已知的臉部，請使用 DetectFaces。要搜尋儲存的影片中已知的臉部，請使用 StartFaceDetection。要搜尋串流視訊中已知的臉部，請使用 CreateStreamProcessor。

人物路徑

Amazon Rekognition 可以追蹤已儲存視訊中偵測到的人員路徑。Amazon Rekognition Video 提供追蹤、臉部詳細資訊、以及在視訊中偵測到人物所在的影格位置資訊。如需詳細資訊，請參閱 人物路徑。

若要偵測儲存的影片中的人物，請使用 StartPersonTracking。

個人防護裝備

Amazon Rekognition 可以偵測到映像中偵測到的人員佩戴的個人防護裝備 (PPE)。Amazon Rekognition 可偵測到面罩、手套和頭套。Amazon Rekognition 預測個人防護裝備的項目是否涵蓋了適當的身體部位。您也可以為偵測到的人員和個人防護裝置物品取得邊界框。如需詳細資訊，請參閱 偵測個人防護裝備。

要檢測圖像中的 PPE，請使用DetectProtectiveEquipment。

名人

Amazon Rekognition 可以在映像與已儲存視訊中辨識數千位名人。您可以取得名人臉孔在映像中的位置、臉部特徵點以及名人的臉部姿態等相關資訊。您可以獲得名人在儲存的影片中出現的追蹤資訊。您

還可以獲得有關公認名人的更多信息，例如表達的情感以及性別的表達方式。如需詳細資訊，請參閱辨識名人。

若要辨識影像中的名人，請使用 RecognizeCelebrities。若要辨識儲存的影片中的名人，請使用 StartCelebrityRecognition。

文字偵測

Amazon Rekognition 映像文字偵測可以偵測映像中的文字並將其轉換為機器可讀取的文字。如需詳細資訊，請參閱 偵測文字。

若要偵測影像中的文字，請使用 DetectText。

不當或冒犯性的內容

Amazon Rekognition 可以分析映像和已儲存的影片中是否含有成人和暴力的內容。如需詳細資訊，請參閱 管制內容。

如要偵測不安全的影像，請使用 DetectModerationLabels。若要偵測不安全的已儲存影片，請使用 StartContentModeration。

自訂

Rekognition 提供的某些映像分析 API 可讓您建立根據自有資料進行訓練的自訂轉接器，藉此提升深度學習模型的準確性。轉接器是外掛於 Rekognition 預先訓練的深度學習模型的元件，可根據您的映像運用領域知識提升其準確性。您可以透過提供和註解範例映像來訓練轉接器以符合您的需求。

建立介面卡之後，系統會提供一個 AdapterId. 您可以將其提供 AdapterId 給作業，以指定要使用您所建立的轉接器。例如，您提供給 DetectModerationLabelsAPI AdapterId 以進行同步影像分析。提供作 AdapterId 為請求的一部分，Rekognition 將自動使用它來增強圖像的預測。這使您可以利用 Rekognition 的功能，同時對其進行自定義以滿足您的需求。

您還可以選擇使用 StartMediaAnalysisJobAPI 批量獲取圖像的預測。如需詳細資訊，請參閱批量分析。

您可以將映像上傳至 Rekognition 主控台並對這些映像執行分析，以評估 Rekognition 操作的準確性。Rekognition 會使用選取的特徵為您的映像加上註解，然後您可以使用經過驗證的預測來檢閱預測，以判斷哪些標籤可以從建立轉接器中受益。

目前您可以將介面卡與 DetectModerationLabels. 如需有關建立和使用轉接器的詳細資訊，請參閱 透過自訂管制提升準確性。

批量分析

Rekognition 大量分析可讓您使用資訊清單檔案和作業，以非同步方式處理大量影像集合。StartMediaAnalysisJob如需詳細資訊，請參閱 批量分析。

映像和影片操作

Amazon Rekognition 提供兩個用於影像和視訊分析的主要 API 集：

- Amazon Rekognition 圖像：此 API 是專為分析圖像而設計的。
- Amazon Rekognition Video：此 API 著重於分析存儲和流式傳輸視頻。

這兩個 API 都可以檢測各種實體，例如面和對象。如需全面瞭解支援的比較和偵測類型，請參閱（詳見）一節分析類型。

Amazon Rekognition Image 操作

Amazon Rekognition 映像作業是同步的。輸入與回應為 JSON 格式。Amazon Rekognition Image 映像操作用於分析 .jpg 或 .png 映像格式的輸入映像檔。傳遞到 Amazon Rekognition Image 操作的映像可儲存於 Amazon S3 儲存貯體中。如果您沒有使用 AWS CLI，也可以將 Base64 編碼映像位元組直接傳遞至 Amazon Rekognition 操作。如需詳細資訊，請參閱使用影像。

Amazon Rekognition Video 操作

亞馬遜重新認知影片 API 有助於分析存放在 Amazon S3 儲存貯體或透過 Amazon Kinesis Video Streams 片。

對於儲存的視訊操作，請注意下列事項：

- 操作是異步的。
- 必須使用「開始」操作來啟動分析（例如，對StartFaceDetection於存儲的視頻中的臉部檢測）。
- 分析的完成狀態會發佈至 Amazon SNS 主題。
- 若要擷取分析結果，請使用對應的「取得」作業 (例如 GetFaceDetection)。
- 如需詳細資訊，請參閱使用儲存的視訊分析。

對於串流視訊分析：

- 功能包括 Rekognition 視訊集合中的臉部搜尋和標籤 (物件或概念) 偵測。
- 標籤的分析結果會以 Amazon SNS 和 Amazon S3 通知的形式傳送。
- 臉部搜尋結果會輸出至 Kinesis 資料串流。
- 串流視訊分析的管理是透過 Amazon Rekognition Video 串流處理器完成的 (例如，使用建立處理器)。CreateStreamProcessor
- 如需詳細資訊，請參閱使用串流視訊事件。

每個視訊分析作業都會傳回正在分析之視訊的相關中繼資料，以及工作 ID 和工作標籤。視訊的「標籤偵測」和「內容協調」等作業可依時間戳記或標籤名稱排序，並依時間戳記或區段彙總結果。

非儲存與以儲存為基礎的操作

Amazon Rekognition 操作將依組分為以下類別。

- 非儲存 API 操作：在這些操作中，Amazon Rekognition 將不會保留任何資訊。您提供輸入映像與影片，操作將執行分析並傳回結果，但 Amazon Rekognition 不會儲存任何內容。如需詳細資訊，請參閱 非儲存操作。
- 以儲存為基礎的 API 操作：Amazon Rekognition 伺服器可將偵測到的臉部資訊儲存在容器中，也就是集合。Amazon Rekognition 提供額外的 API 操作，您可使用這些操作來搜尋保留的臉部資訊並尋找臉部配對。如需詳細資訊，請參閱 以儲存為基礎的 API 操作。

使用 AWS 開發套件或 HTTP 來呼叫 Amazon Rekognition API 操作

您可以使用 AWS 開發套件或直接使用 HTTP 來呼叫 Amazon Rekognition API 操作。除非有充分理由不使用 AWS 開發套件，否則應一律使用 AWS 開發套件。本節中的 Java 範例使用 AWS 開發套件。未提供 Java 專案檔案，但您可以使用 AWS Toolkit for Eclipse 來使用 Java 開發 AWS 應用程式。

本節中的 .NET 範例使用 AWS SDK for .NET。您可以使用 AWS Toolkit for Visual Studio 使用 .NET 開發 AWS 應用程式。它包含用於部署應用程式和管理服務的有用範本和 AWS Explorer。

本指南中的 API 參考 涵蓋使用 HTTP 呼叫 Amazon Rekognition 操作的方法。如需 Java 參考資訊，請參閱 AWS SDK for Java。

您可以使用的 Amazon Rekognition 服務端點記錄於 AWS 區域與端點。

以 HTTP 呼叫 Amazon Rekognition 時，請使用 POST HTTP 操作。

非儲存與以儲存為基礎的 API 操作

Amazon Rekognition 提供兩種類型的 API 操作。其中一種是非儲存操作，Amazon Rekognition 不會儲存資訊；另一種是儲存操作，Amazon Rekognition 會儲存特定臉部資訊。

非儲存操作

Amazon Rekognition 提供以下用於映像的非儲存 API 操作：

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

Amazon Rekognition 提供以下用於影片的非儲存 API 操作：

- StartLabelDetection
- StartFaceDetection
- StartPersonTracking
- StartCelebrityRecognition
- StartContentModeration

這些稱為非儲存 API 操作，因為當您執行操作呼叫時，Amazon Rekognition 不會保留任何從輸入映像中找到的任何相關資訊。與所有其他 Amazon Rekognition API 操作相同，非儲存 API 操作不會保留輸入映像位元組。

以下範例說明您在應用程式中可能採用非儲存 API 操作的情況。這些情況假設您有一個存放映像的本機儲存庫。

Example 1：用於在包含特定標籤的本機儲存庫中尋找映像的應用程式

首先，在儲存庫內的每個映像中使用 Amazon Rekognition `DetectLabels` 操作來偵測標籤，並建置客戶端索引，如下所示：

```
Label        ImageID

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

接著，您的應用程式可以搜尋這個索引來尋找在本機儲存庫中包含特定標籤的映像。例如，顯示出現樹的映像。

每個 Amazon Rekognition 偵測的標籤都有相關的可信度數值。此分數顯示輸入映像內包含該標籤的可信度等級。您可以使用此可信度分數來選擇是否根據應用程式要求的偵測可信度等級來執行額外的客戶端標籤篩選。例如，如果您要求精確標籤，可篩選並選擇有較高可信度分數的標籤 (例如，95% 或更高分數)。如果您的應用程式未要求較高的可信度分數，您可以選擇篩選出有較低可信度分數的標籤 (接近於 50%)。

Example 2：顯示強化臉部映像的應用程式

首先，可使用 Amazon Rekognition `DetectFaces` 操作來偵測本機儲存庫內每個映像中的臉部，並建置客戶端索引，如下所示：操作將傳回每個臉部的中繼資料，包含週框方塊、臉部特徵點 (例如嘴巴與耳朵的位置)、以及臉部屬性 (例如，性別)。您可以將此中繼資料儲存於客戶端本機索引，如下所示：

```
ImageID     FaceID     FaceMetaData

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

在此索引中，主索引鑰為 `ImageID` 和 `FaceID` 的組合。

接著，您可以使用索引中的資訊，在應用程式顯示來自本機儲存庫的映像時用以強化映像。例如，您可以加入臉部周圍的週框方塊或者強調臉部特徵。

以儲存為基礎的 API 操作

Amazon Rekognition 影像支援此IndexFaces作業，您可以使用該作業偵測影像中的臉部，並保留 Amazon Rekognition 集合中偵測到的臉部特徵相關資訊。此為以儲存體為基礎的 API 操作範例，因服務將在伺服器中保留資訊。

Amazon Rekognition Image 提供以下儲存 API 操作：

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

Amazon Rekognition Video 提供以下儲存 API 操作：

- StartFaceSearch
- CreateStreamProcessor

若要儲存臉部資訊，您必須先在帳戶中其中一個 AWS 區域中建立臉部集合。當您呼叫 `IndexFaces` 操作時可指定此臉部集合。在建立臉孔集合並儲存所有臉孔的臉部特徵資訊後，即可搜尋集合來比對臉孔。例如，您可以呼叫 `searchFacesByImage.` 來偵測映像中最大的臉孔並在集合中搜尋相符的臉部

Amazon Rekognition Video 操作可存取 `IndexFaces` 儲存於集合中的臉部資訊。例如，您可以呼叫 StartFaceSearch 來搜尋影片中符合現有集合中之臉孔的人物。

如需建立和管理集合的詳細資訊，請參閱 在集合中搜尋人臉。

> **Note**
>
> 集合存儲面向量，這是面的數學表示法。集合不會儲存臉部影像。

Example 1：建築物門禁驗證應用程式

首先需建立一個臉部集合，使用 IndexFaces 操作將掃描的名牌映像儲存於該集合，而此操作可擷取臉部並儲存為可搜尋的映像向量。接著，當員工進入大樓時，將拍攝員工臉部的映像並傳送至 SearchFacesByImage 操作。如果臉部比對的相似性分數達到設定的目標 (例如 99%)，即可驗證該員工身分。

模型版本控制

Amazon Rekognition 使用深度學習模型來執行臉部偵測並在集合中搜尋臉孔。持續根據客戶的意見回饋來改善模型的精確度，並推動深入學習研究領域。這些改善功能將隨模型更新提供。例如，在模型版本 1.0 中，IndexFaces 可將影像中前 15 個最大的臉部編入索引。模型的較新版本則可讓 IndexFaces 將映像中前 100 個最大的臉部編入索引。

當您建立新的集合時，將採用最新版的模型版本。為提高精確度，模型將不定期更新。

當有新版本的模型發行時，會發生下列情況：

- 您建立的新集合將採用最新模型。您使用 IndexFaces 加入至新集合中的臉孔將以最新模型進行偵測。
- 您現有的集合繼續使用建立時採用的模型版本。儲存在這些集合中的臉部向量將不會自動更新為最新模型版本。
- 新增到現有集合中的新臉孔將使用該集合原先使用的模型進行偵測。

不同的模型版本彼此無法相容。特別是當映像編入多個集合的索引，而這些集合皆採用不同版本的模型時，同一個偵測到的臉部會有不同的臉部識別碼。若映像編入多個集合的索引，而這些集合使用相同模型時，臉部識別碼將會相同。

若您的集合管理不負責模型的更新，應用程式可能遇到相容性問題。您可以使用為了回應集合操作所傳回的 FaceModelVersion 欄位 (例如 CreateCollection) 來判斷目前採用的模型版本。您可以透過呼叫，取得現有集合的模型版本 DescribeCollection。如需詳細資訊，請參閱 描述集合。

集合中現有的臉部向量無法更新為較新的模型版本。由於 Amazon Rekognition 不會儲存原始映像位元組，因此將無法自動使用較新的模型版本來重新編制映像索引。

若要在現有集合中儲存的臉部使用最新模型，請建立新的集合 (CreateCollection) 並重新將原始影像編入新集合索引 (Indexfaces)。您需要更新任何由應用程式儲存的臉部識別碼，因為新集合中的臉部識別碼與舊集合中的臉部識別碼不同。若您不再需要因為舊集合，可使用 DeleteCollection 來刪除。

無狀態操作 (例如 DetectFaces) 會使用最新版本的模型。

Amazon Rekognition 入門

本節主題可用於您的 Amazon Rekognition 入門操作。如果您是 Amazon Rekognition 新手，建議您先回顧 Amazon Rekognition 的運作方式 中的概念與術語。

您需要先建立 AWS 帳戶並取得 AWS 帳戶 ID，才可以使用 Rekognition。您也需要建立一個使用者，讓 Amazon Rekognition 系統判斷您是否具有存取其資源所需的許可。

建立帳戶之後，您需要安裝和設定 AWS CLI 和 AWS SDK。 AWS CLI 可讓您透過命令列與 Amazon Rekognition 和其他服務互動，而 AWS 開發套件可讓您使用程式設計語言 (例如 Java 和 Python) 與 Amazon Rekognition 進行互動。

設定好 AWS CLI 和 AWS SDK 之後，您可以查看一些如何使用它們的範例。您也可以檢視一些如何使用主控台與 Amazon Rekognition 互動的範例。

主題

步驟 1：設定 AWS 帳戶並建立使用者

第一次使用 Amazon Rekognition 之前，您必須完成下列任務：

1. 註冊一個 AWS 帳戶。
2. 建立使用者。

開發人員指南的這一部分解釋了您將建立 AWS 帳戶和使用者的原因以及方式。

主題

創建一個 AWS 帳戶和用戶

AWS 帳戶

註冊 Amazon Web Services (AWS) 時，您的 AWS 帳戶會自動註冊 AWS 的所有服務，包括 Amazon Rekognition。您只需針對所使用的服務付費。

使用 Amazon Rekognition 時，您僅需按使用的資源量付費。

如果您是新 AWS 客戶，您可以免費開始使用 Amazon Rekognition。如需詳細資訊，請參閱 AWS 免費用量方案。

有關帳戶建立說明，請參閱後續的 註冊一個 AWS 帳戶 章節。

如果您已有 AWS 帳戶，請略過帳戶設定並建立系統管理使用者。

使用者

AWS 服務，例如 Amazon Rekognition，會在您進行存取時，要求您提供憑證。這樣服務可以確定您是否有權存取該服務所擁有的資源。

您可以為 AWS 帳戶建立存取金鑰以存取 AWS CLI 或 API，而使用主控台需要您的密碼。不過，不建議您透過使用 AWS 帳戶根使用者的憑證來存取 AWS。相反地，我們建議您使用 AWS Identity and Access Management (IAM) 建立管理使用者。

然後您就可以使用特殊的 URL 與該管理使用者的憑證存取 AWS。

如果您已註冊 AWS，但是尚未為自己建立使用者，可以透過使用 IAM 主控台加以建立。如需有關如何建立系統管理使用者的指示，請參閱後續 建立具有管理權限的使用者 章節。

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。
2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱 AWS IAM Identity Center 使用者指南中的啟用 AWS IAM Identity Center。
2. 在 IAM 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM 身分中心使用者登入的說明，請參閱使用AWS 登入 者指南中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。

2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「新增群組」。

步驟 2：設定 AWS CLI 和開 AWS 發套件

主題

- 授與程式設計存取權
- 搭配 SDK 使用 Rekognition AWS

下列步驟說明如何安裝本文件中範例所使用的 AWS Command Line Interface (AWS CLI) 和 AWS SDK。有許多不同的方式來驗證 AWS SDK 呼叫。本指南中的範例假設您使用預設認證設定檔來呼叫 AWS CLI 命令和 AWS SDK API 作業。

如需可用 AWS 區域的清單，請參閱中的區域和端點Amazon Web Services 一般參考。

請按照以下步驟下載和配置 AWS SDK。

若要設定 AWS CLI 和 AWS 軟體開發套件

1. 下載AWS CLI並安裝您要使用的和 AWS SDK。本指南提供了 Java AWS CLI、Python、紅寶石、Node.js、PHP、 JavaScript. 如需安裝 AWS 開發套件的相關資訊，請參閱 Amazon Web Services 的工具。
2. 為您在 創建一個 AWS 帳戶和用戶 中建立的使用者建立存取金鑰。

   a. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。

   b. 在導覽窗格中，選擇使用者 。

   c. 選擇您在 創建一個 AWS 帳戶和用戶 中建立之使用者選擇名稱。

   d. 選擇 Security credentials (安全憑證) 索引標籤。

   e. 選擇 Create access key (建立新的存取金鑰)。然後選擇 Download .csv file (下載 .csv 檔案)，將存取金鑰 ID 和私密存取金鑰儲存至電腦上的 CSV 檔案。將檔案存放在安全位置。在關閉此對話方塊後，您將無法再次存取該私密存取金鑰。在您下載該 CSV 檔案後，選擇 Close (關閉)。

3. 如果您已安裝 AWS CLI，則可以aws configure在命令提示字元中輸入，為大多數 AWS SDK 設定認證和區域。否則，請使用下列說明。
4. 在您的電腦上，瀏覽至主目錄並建立 .aws 目錄。在 Unix 系統上 (例如 Linux 或 macOS)，其會是下列位置：

   ```
   ~/.aws
   ```

   在 Windows 上，其會是下列位置：

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

5. 在 .aws 目錄中，建立稱為 credentials 的新檔案。
6. 開啟您在步驟 2 下載的憑證 CSV 檔案，並使用下列格式，將其內容複製到 credentials 檔案：

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

   以 your_access_key_id 與 your_secret_access_key 替換您的存取金鑰 ID 和私密存取金鑰。
7. 儲存 Credentials 檔案，並刪除該 CSV 檔案。
8. 在 .aws 目錄中，建立稱為 config 的新檔案。
9. 開啟 config 檔案，並使用下列格式，輸入您的區域。

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

   將您所需要的 AWS 區域 (例如，us-west-2) 替換為 your_aws_region。

   > **Note**
   >
   > 如果您不選擇區域，則預設使用 us-east-1。

10. 儲存 config 檔案。

授與程式設計存取權

您可以在本機電腦或其他 AWS 環境 (例如 Amazon 彈性運算雲端執行個體) 上執行本指南中的 AWS CLI 和程式碼範例。若要執行範例，您必須授與範例所使用之 AWS SDK 作業的存取權。

主題

在本機電腦執行程式碼

若要在本機電腦上執行程式碼，建議您使用短期認證授與使用者存取 AWS SDK 作業。如需有關在本機電腦上執行 AWS CLI 和程式碼範例的特定資訊，請參閱在本機電腦上使用設定檔。

如果使用者想要與 AWS 之外的 AWS Management Console. 授與程式設計存 AWS取權的方式取決於正在存取的使用者類型。

若要授與使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	By
人力身分 (IAM Identity Center 中管理的使用者)	使用臨時登入資料來簽署對 AWS CLI、 AWS SDK 或 AWS API 的程式設計要求。	請依照您要使用的介面所提供的指示操作。 • 如需詳細資訊 AWS CLI，請參閱《使 AWS CLI 用AWS Command Line Interface者指南》 AWS IAM Identity Center中的〈配置使用〉。 • 如需 AWS SDK、工具和 AWS API，請參閱 AWS SDK 和工具參考指南中的 IAM 身分中心身分驗證。
IAM	使用臨時登入資料來簽署對 AWS CLI、 AWS SDK 或 AWS API 的程式設計要求。	遵循《IAM 使用者指南》中的〈將臨時登入資料搭配 AWS 資源使用〉中的指示

哪個使用者需要程式設計存取權？	到	By
IAM	(不建議使用) 使用長期認證來簽署對 AWS CLI、 AWS SDK 或 AWS API 的程式設計要求。	請依照您要使用的介面所提供的指示操作。 • 如需相關資訊 AWS CLI，請參閱使用指南中的使用 IAM 使用者登入資料進行驗證。AWS Command Line Interface • 對於 AWS SDK 和工具，請參閱 AWS SDK 和工具參考指南中的使用長期憑據進行身份驗證。 • 如需 AWS API，請參閱 IAM 使用者指南中的管理 IAM 使用者的存取金鑰。

在本機電腦上使用設定檔

您可以使用您在中建立的短期認證來執行本指南中的 AWS CLI 和程式碼範例在本機電腦執行程式碼。若要取得認證和其他設定資訊，範例會使用名為 `profile-name` 的設定檔，例如：

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

設定檔所代表的使用者必須具有呼叫 Rekognition SDK 作業的權限，以及範例所需的其他 AWS SDK 作業。

若要建立可搭配 AWS CLI 和程式碼範例使用的描述檔，請選擇下列其中一項。請確定您建立的設定檔名稱是 `profile-name`。

- 由 IAM 管理的使用者 — 請按照 切換到 IAM 角色 (AWS CLI) 中的指示進行操作。
- 員工身分 (由管理的使用者 AWS IAM Identity Center) — 按照設定 AWS CLI 中的說明進行操作 AWS IAM Identity Center。對於程式碼範例，我們建議使用整合式開發環境 (IDE)，該環境支援透過 IAM Identity Center 啟用身分驗證的 AWS 工具組。如需 Java 範例，請參閱 使用 Java 開始建置。

如需 Python 範例，請參閱 使用 Python 開始建置。如需更多詳細資訊，請參閱 什麼是 IAM Identity Center 憑證。

> **Note**
>
> 您可以使用程式碼取得短期憑證。如需更多相關資訊，請參閱 切換到 IAM 角色 (AWS API)。對於 IAM Identity Center，請遵循 取得 CLI 存取的 IAM 角色憑證 中的指示，獲得某個角色的短期憑證。

在 AWS 環境中執行程式碼

您不應該使用用戶憑據在 AWS 環境中簽署 AWS SDK 調用，例如在 AWS Lambda 函數中運行的生產代碼。相反地，您可以一個角色來定義您的程式碼所需的權限。然後，您可以將該角色附加到程式碼執行的環境。如何附加角色並提供臨時憑證取決於程式碼執行的環境：

- AWS Lambda 函數 — 當 Lambda 擔任 Lambda 函數的執行角色時，請使用 Lambda 自動提供給函數的臨時登入資料。這些憑證可在 Lambda 環境變數中使用。您不需要指定設定檔。如需更多詳細資訊，請參閱 Lambda 執行角色。
- Amazon EC2 — 使用 Amazon EC2 執行個體中繼資料端點憑證提供者。提供者會使用您附加到 Amazon EC2 執行個體的 Amazon EC2 執行個體設定檔，為您自動產生和重新整理憑證。如需更多詳細資訊，請參閱 使用 IAM 角色向在 Amazon EC2 執行個體上執行的應用程式授予權限。
- Amazon Elastic Container Service — 使用容器憑證提供者。Amazon ECS 會將憑證傳送並重新整理到中繼資料端點。您指定的 任務 IAM 角色 提供了用於管理應用程式使用的憑證的策略。如需更多詳細資訊，請參閱 與 AWS 服務互動。

如需憑證提供者的詳細資訊，請參閱標準化憑證提供者。

搭配 SDK 使用 Rekognition AWS

AWS 軟件開發套件（SDK）可用於許多流行的編程語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

SDK 文件	代碼範例
AWS SDK for C++	AWS SDK for C++ 程式碼範例

SDK 文件	代碼範例
AWS CLI	AWS CLI 程式碼範例
AWS SDK for Go	AWS SDK for Go 程式碼範例
AWS SDK for Java	AWS SDK for Java 程式碼範例
AWS SDK for JavaScript	AWS SDK for JavaScript 程式碼範例
適用於 Kotlin 的 AWS SDK	適用於 Kotlin 的 AWS SDK 程式碼範例
AWS SDK for .NET	AWS SDK for .NET 程式碼範例
AWS SDK for PHP	AWS SDK for PHP 程式碼範例
AWS Tools for PowerShell	PowerShell 程式碼範例的工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 程式碼範例
AWS SDK for Ruby	AWS SDK for Ruby 程式碼範例
適用於 Rust 的 AWS SDK	適用於 Rust 的 AWS SDK 程式碼範例
適用於 SAP ABAP 的 AWS SDK	適用於 SAP ABAP 的 AWS SDK 程式碼範例
適用於 Swift 的 AWS SDK	適用於 Swift 的 AWS SDK 程式碼範例

如需 Rekognition 專屬範例，請參閱 使用 SDK 的 Amazon Rekognition 的程式碼範例 AWS。

> **可用性範例**
>
> 找不到所需的內容嗎？ 請使用本頁面底部的提供意見回饋連結申請程式碼範例。

步驟 3：開始使用 AWS CLI 和 AWS SDK API

設定好要使用的 AWS CLI 和 AWS 開發套件之後，您可以建置使用 Amazon Rekognition 的應用程式。下列主題說明如何開始使用 Amazon Rekognition Image 和 Amazon Rekognition Video。

- 使用映像
- 使用儲存的影片分析
- 使用串流視訊事件

格式化範 AWS CLI 例

本指南中的 AWS CLI 範例針對 Linux 作業系統進行格式化。若要搭配 Microsoft Windows 使用範例，您需要變更 `--image` 參數的 JSON 格式，並將換行符號從反斜線 (\) 變更為插入號 (^)。如需 JSON 格式的詳細資訊，請參閱指定 AWS 命令列界面的參數值。

以下是為 Microsoft Windows 格式化的示例 AWS CLI 命令（請注意，這些命令不會按原樣運行，它們只是格式化示例）：

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

您也可以提供同時適用於 Microsoft Windows 與 Linux 的 JSON 速記版本。

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

如需詳細資訊，請參閱搭配 AWS 命令列界面使用速記語法。

下一步驟

步驟 4：開始使用 Amazon Rekognition 主控台

步驟 4：開始使用 Amazon Rekognition 主控台

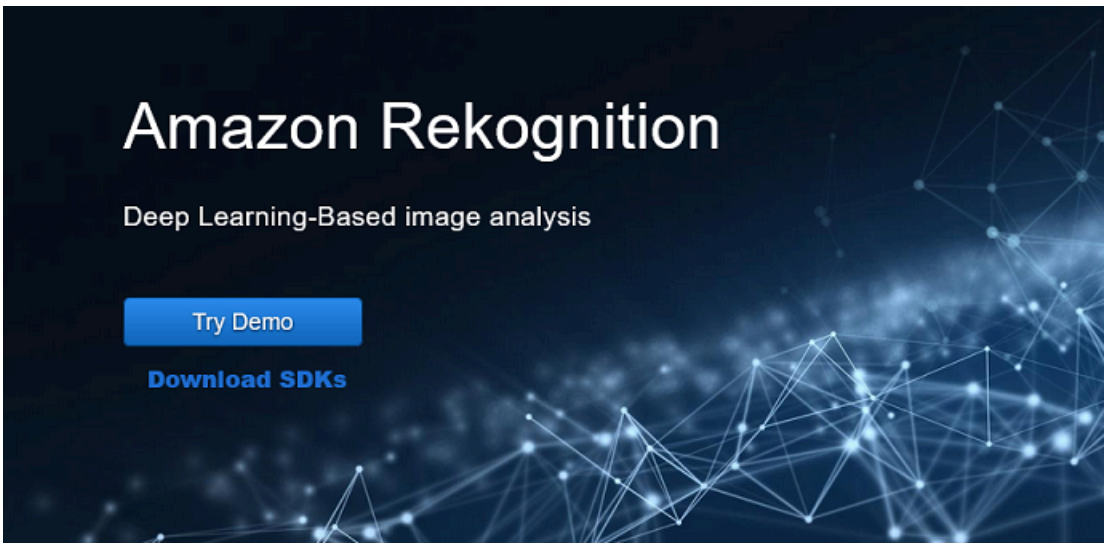
本節說明如何使用 Amazon Rekognition 的部分功能，例如物件與場景偵測、人臉分析以及在一組影像中執行人臉比對。如需詳細資訊，請參閱 Amazon Rekognition 的運作方式。您也可以使用 Amazon Rekognition API 或 AWS CLI 偵測物件和場景、偵測臉孔，以及比較和搜尋臉孔。如需詳細資訊，請參閱 步驟 3：開始使用 AWS CLI 和 AWS SDK API。

本節也說明如何使用 Rekognition 主控台查看 Rekognition 的彙總 Amazon CloudWatch 指標。

主題

- 設定主控台權限

- 練習 1：偵測物件和場景 (主控台)
- 練習 2：分析影像中的人臉 (主控台)
- 練習 3：比對影像中的人臉 (主控台)
- 練習 4：查看彙總指標 (主控台)



設定主控台權限

若要使用 Rekognition 主控台，您必須擁有存取主控台之角色或帳戶的適當權限。對於某些操作，Rekognition 會自動建立 Amazon S3 儲存貯體來存放操作期間處理的檔案。如果您想要將訓練檔案儲存在此主控台儲存貯體以外的儲存貯體中，則需要其他權限。

允許存取主控台

若要使用 Rekognition 主控台，您可以使用類似下列的 IAM 政策，其涵蓋 Amazon S3 和 Rekognition 主控台。如需有關指派權限的資訊，請參閱指派權限。

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

```
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleS3BucketFirstUseSetupAccess",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketVersioning",
                "s3:PutLifecycleConfiguration",
                "s3:PutEncryptionConfiguration",
                "s3:PutBucketPublicAccessBlock",
                "s3:PutCors",
                "s3:GetCors"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*"
        },
        {
            "Sid": "RekognitionConsoleS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*"
        },
        {
            "Sid": "RekognitionConsoleS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:HeadObject",
                "s3:DeleteObject",
```

```
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*/*"
        },
        {
            "Sid": "RekognitionConsoleManifestAccess",
            "Effect": "Allow",
            "Action": [
                "groundtruthlabeling:*",
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleTagSelectorAccess",
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleKmsKeySelectorAccess",
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        }
    ]
}
```

存取外部 Amazon S3 儲存貯體

當您第一次在新的 AWS 區域中開啟 Rekognition 主控台時，Rekognition 會建立用來儲存專案檔案的值區 (主控台值區)。或者，您可以使用自己的 Amazon S3 儲存貯體 (外部儲存貯體) 將圖像或清單檔案上傳到主控台。若要使用外部儲存貯體，請將下列政策區塊新增至前述政策。以您的儲存貯體名稱取代 my-bucket。

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

指派許可

若要提供存取權，請新增權限至您的使用者、群組或角色：

- AWS IAM Identity Center (AWS Single Sign-On 的後續產品) 中的使用者和群組：

  建立權限合集。遵循《AWS IAM Identity Center (AWS Single Sign-On 的後續產品) 使用者指南》中建立權限集中的指示。

- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。

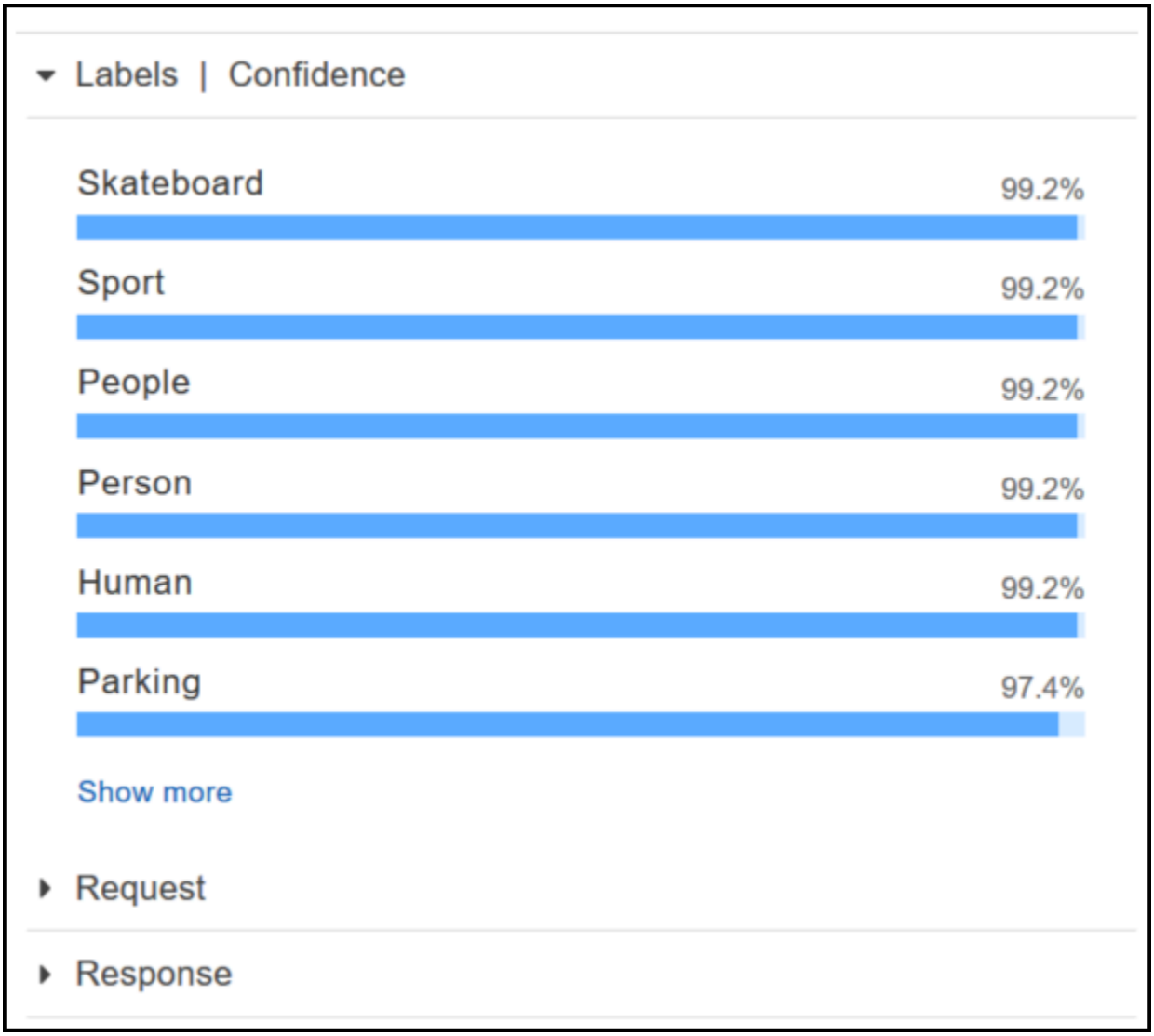
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照 IAM 使用者指南 的 為 IAM 使用者建立角色 中的指示進行操作。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台)中的指示。

練習 1：偵測物件和場景 (主控台)

此節將从高層面具體說明 Amazon Rekognition 的物件與場景偵測功能的運作方式。當您指定影像做為輸入檔時，服務會偵測影像中的物件和場景，並加上每個物件與場景的可信度分數百分比後傳回結果。

例如，Amazon Rekognition 將偵測範例影像中的下列物件與場景：滑板、運動、人物、汽车、轎車與其他車輛。

Amazon Rekognition 也將針對範例影像中偵測到的物件傳回可信度分數，如下列範例回應所示。



若要查看所有顯示於此回應中的可信度分數，請選擇標籤|可信度中的顯示更多窗格。

您也可以查看對 API 的請求以及來自 API 的回應做為參考。

請求

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

```
         }
      }
   }
}
```

回應

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

```
            "Name":"Car"
         },
         {
            "Confidence":91.53300476074219,
            "Name":"Vehicle"
         },
         {
            "Confidence":76.85114288330078,
            "Name":"Intersection"
         },
         {
            "Confidence":76.85114288330078,
            "Name":"Road"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Boardwalk"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Path"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Pavement"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Sidewalk"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Walkway"
         },
         {
            "Confidence":66.71541595458984,
            "Name":"Building"
         },
         {
            "Confidence":62.04711151123047,
            "Name":"Coupe"
         },
         {
            "Confidence":62.04711151123047,
```

```
            "Name":"Sports Car"
         },
         {
            "Confidence":61.98909378051758,
            "Name":"City"
         },
         {
            "Confidence":61.98909378051758,
            "Name":"Downtown"
         },
         {
            "Confidence":61.98909378051758,
            "Name":"Urban"
         },
         {
            "Confidence":60.978023529052734,
            "Name":"Neighborhood"
         },
         {
            "Confidence":60.978023529052734,
            "Name":"Town"
         },
         {
            "Confidence":59.22066116333008,
            "Name":"Sedan"
         },
         {
            "Confidence":56.48063278198242,
            "Name":"Street"
         },
         {
            "Confidence":54.235477447509766,
            "Name":"Housing"
         },
         {
            "Confidence":53.85226058959961,
            "Name":"Metropolis"
         },
         {
            "Confidence":52.001792907714844,
            "Name":"Office Building"
         },
         {
            "Confidence":51.325313568115234,
```

```
            "Name":"Suv"
         },
         {
            "Confidence":51.26075744628906,
            "Name":"Apartment Building"
         },
         {
            "Confidence":51.26075744628906,
            "Name":"High Rise"
         },
         {
            "Confidence":50.68067932128906,
            "Name":"Pedestrian"
         },
         {
            "Confidence":50.59548568725586,
            "Name":"Freeway"
         },
         {
            "Confidence":50.568580627441406,
            "Name":"Bumper"
         }
      ]
}
```


如需詳細資訊，請參閱 Amazon Rekognition 的運作方式。

在您提供的影像中偵測物件和場景

您可以上傳您擁有的影像或提供影像的 URL 做為用於 Amazon Rekognition 主控台的輸入檔。Amazon Rekognition 將傳回物件和場景、每個物件的可信度分數以及在您所提供的影像中偵測到的場景。

> **Note**
>
> 影像大小必須少於 5MB，且須為 JPEG 或 PNG 格式。

若要在您提供的影像中偵測物件和場景

1. 前往 https://console.aws.amazon.com/rekognition/ 開啟 Amazon Rekognition 主控台。
2. 選擇標籤偵測。
3. 執行以下任意一項：

- 上傳影像 – 選擇 Upload (上傳)，前往您儲存影像的位置，然後選擇影像。
- 使用 URL – 在文字方塊中輸入 URL，然後選擇 Go (前往)。

4. 在 Labels | Confidence (標籤 | 可信度) 窗格中檢視每個偵測到的標籤之可信度分數。

若要取得更多影像分析選項，請參閱 the section called “使用映像”。

偵測影片中的人物和物件

您可以在 Amazon Rekognition 主控台上傳您提供的影片，做為輸入。Amazon Rekognition 會傳回在影片中偵測到的人員、物件和標籤。

> **Note**
>
> 示範影片長度不得超過一分鐘或大於 30 MB。其必須是 MP4 檔案格式，並使用 H.264 編解碼器進行編碼。

偵測影片中的人物和物件

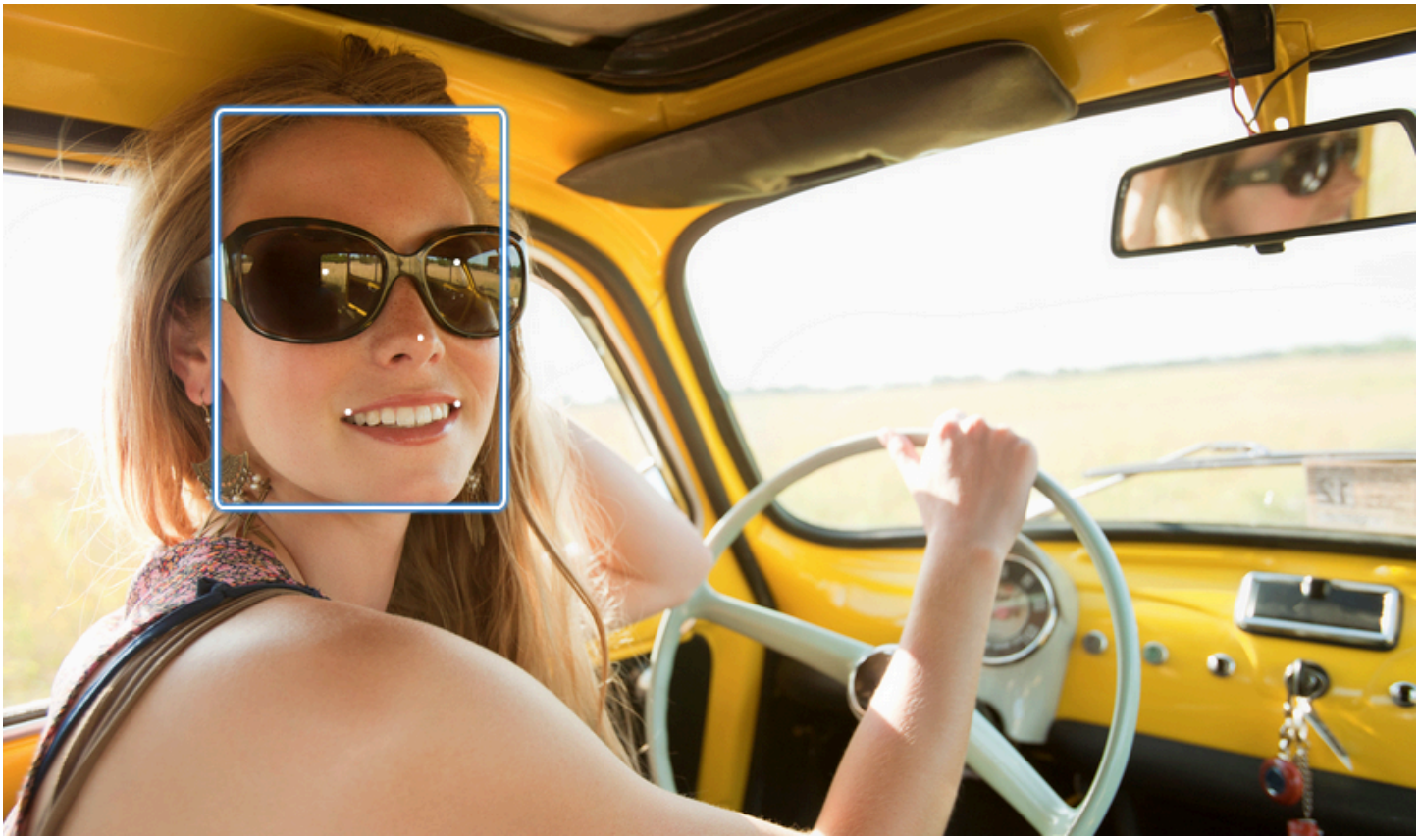
1. 前往 https://console.aws.amazon.com/rekognition/ 開啟 Amazon Rekognition 主控台。
2. 從導覽列中選擇 [儲存的視訊分析]。
3. 在 [選擇樣本或上傳您自己的影片] 下方，從下拉式選單中選取 [您自己的影片]。
4. 拖放影片，或從儲存影片的位置選取影片。

如需更多影片分析選項，請參閱 the section called “使用儲存的影片分析” 或 the section called “使用串流視訊事件”。

練習 2：分析影像中的人臉 (主控台)

本節說明如何使用 Amazon Rekognition 主控台來偵測人臉與分析影像中的人臉屬性。當您提供含有臉部的影像做為輸入檔時，服務會偵測影像中的臉部、分析臉部的臉部屬性，然後傳回影像中偵測到之臉部與臉部屬性的可信度分數百分比。如需詳細資訊，請參閱 Amazon Rekognition 的運作方式。

例如，如果您選擇以下範例影像做為輸入檔，Amazon Rekognition 將該視該影像為人臉進行偵測，並傳回偵測到的人臉與人臉屬性之可信度分數。



下列顯示回應範例。

Results

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

Show less

如果輸入影像中有多個人臉，Rekognition 最多可在影像中偵測 100 個人臉。每一個偵測到的臉部皆以方框標註。當您點擊標註了人臉的方框時，Rekognition 將在人臉|可信度窗格中顯示偵測到的人臉與其屬性之可信度分數。

分析您提供的影像中的人臉

可在 Amazon Rekognition 主控台中上傳您自己的影像或提供影像的 URL。

> **Note**
>
> 影像大小必須少於 5MB，且須為 JPEG 或 PNG 格式。

若要分析您提供的影像中的臉部

1. 前往 https://console.aws.amazon.com/rekognition/ 開啟 Amazon Rekognition 主控台。
2. 選擇 Facial analysis (臉部分析)。
3. 執行以下任意一項：
   - 上傳影像 – 選擇 Upload (上傳)，前往您儲存影像的位置，然後選擇影像。
   - 使用 URL – 在文字方塊中輸入 URL，然後選擇 Go (前往)。
4. 在 Faces | Confidence (臉部 | 可信度) 窗格中檢視其中一個偵測到的臉部可信度分數與其屬性。
5. 如果影像中有多個臉部，請選擇其它臉部來查看屬性與分數。

練習 3：比對影像中的人臉 (主控台)

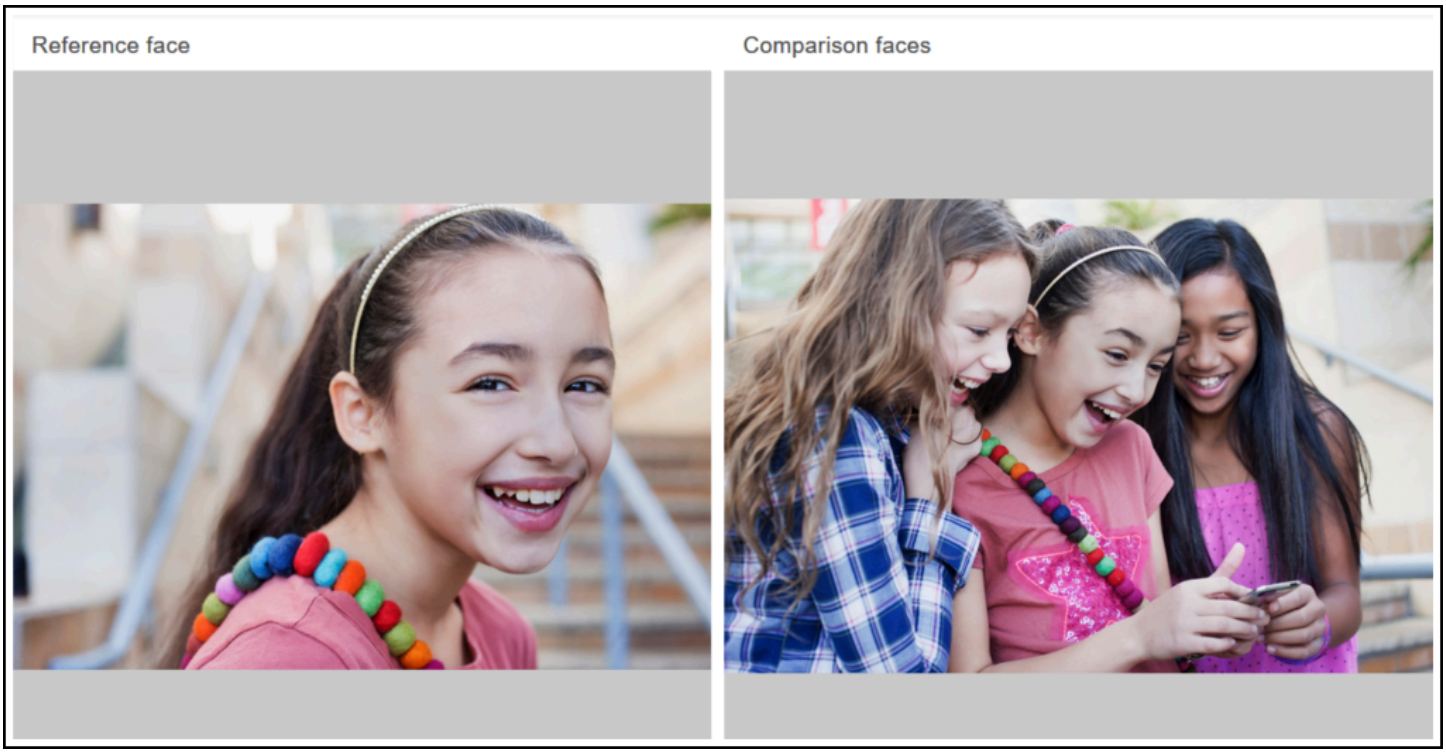
本節說明如何使用 Amazon Rekognition 主控台在一組影像的多個臉孔中進行人臉比對。當您指定參考人臉 (原始) 和對比人臉 (目標) 影像時，Rekognition 將以原始影像 (也就是參考人臉) 中最大的臉孔為基礎來比對目標影像 (也就是對比人臉) 中的臉孔，最多可辨識 100 張臉孔，接著便可判定原始影像中的人臉與目標影像中的人臉之間的相似度。每次比對的相似度分數會顯示在 Results (結果) 窗格中。

如果目標影像內包含多張臉孔，Rekognition 將對原始影像中的臉孔與目標影像中所偵測到的最多 100 個人臉進行比對，然後指定相似度分數給每個比對結果。

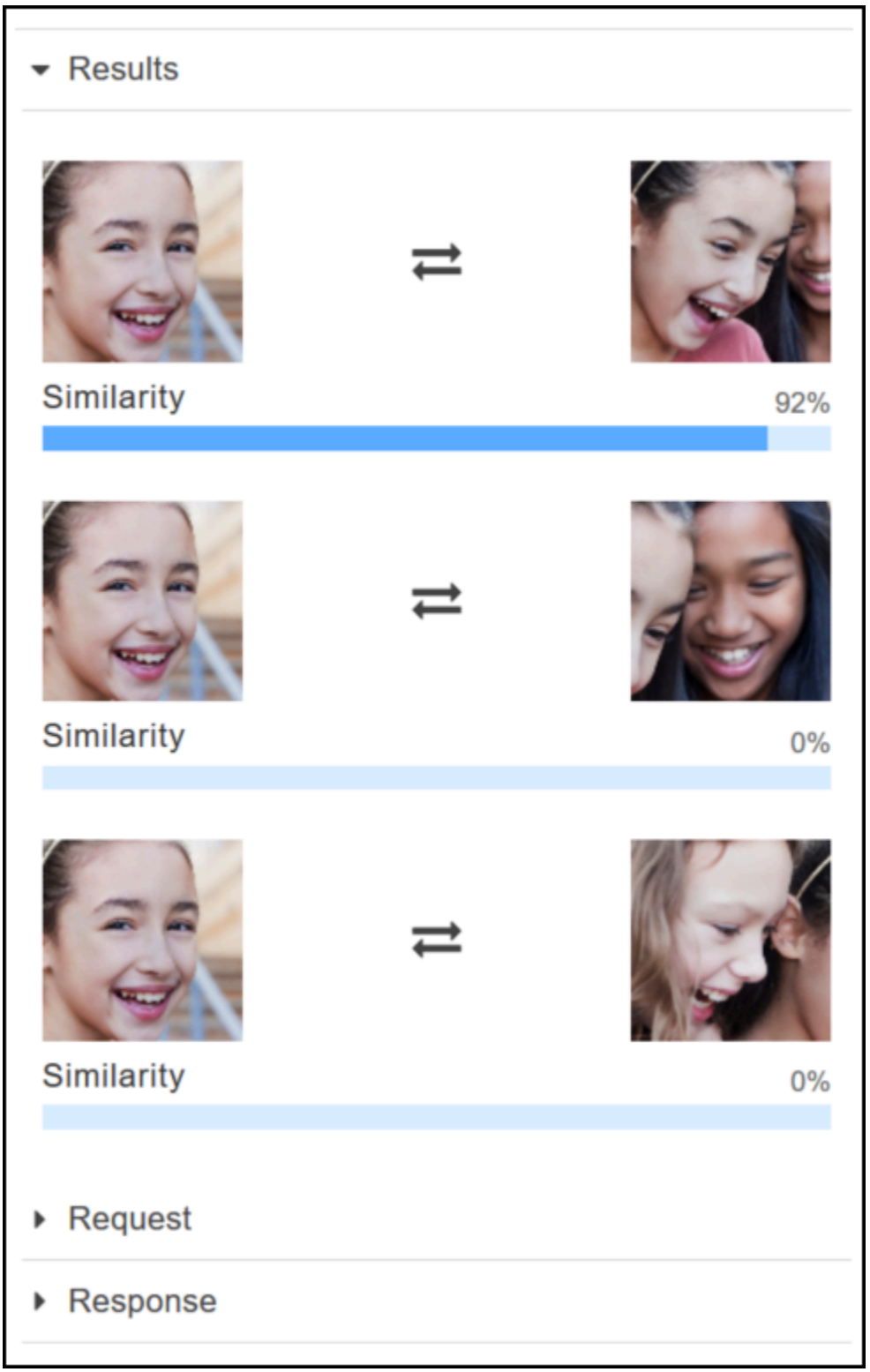
如果原始影像含有多張臉孔，服務將偵測原始影像中最大的臉孔，使用它與目標影像中偵測到的每個臉孔進行比對。

如需詳細資訊，請參閱 比較映像中的人臉。

例如，使用顯示在左側的範例影像做為原始影像、右側的範例影像則做為目標影像，Rekognition 將偵測原始影像中的臉孔並與在目標影像中偵測到的每個臉孔進行比較，最後為每組顯示相似度分數。



以下顯示在目標影像偵測到的臉孔以及每個臉孔的相似度分數。



比較您提供的影像中的人臉

您可以上傳自己的原始和目標影像，供 Rekognition 比對影像中的臉孔，或者您可以指定影像所在的 URL 位置。

> **Note**
>
> 影像大小必須少於 5MB，且須為 JPEG 或 PNG 格式。

若要比較影像中的臉部

1. 前往 https://console.aws.amazon.com/rekognition/ 開啟 Amazon Rekognition 主控台。
2. 選擇 Face comparison (臉部比較)。
3. 指定原始影像可使用以下其中一個方法：
   - 上傳影像 – 選擇左側的 Upload (上傳)，前往您儲存原始影像的位置，然後選擇影像。
   - 使用 URL – 在文字方塊中輸入原始影像的 URL，然後選擇 Go (前往)。
4. 指定目標影像可使用以下其中一個方法：
   - 上傳影像 – 選擇右側的 Upload (上傳)，前往您儲存目標影像的位置，然後選擇影像。
   - 使用 URL – 在文字方塊中輸入原始影像的 URL，然後選擇 Go (前往)。
5. Rekognition 將偵測原始影像中最大的臉孔並與目標影像中最多 100 張臉孔進行比對，接著在結果窗格中顯示每個人臉配對的相似度分數。

練習 4：查看彙總指標 (主控台)

Amazon Rekognition 指標窗格將顯示在一段期間內的 Rekognition 指標彙總活動圖形。例如，`SuccessfulRequestCount` 彙總指標會顯示過去七天內對所有 Rekognition API 操作的成功請求總數。

下表列出顯示在 Rekognition 指標窗格中的圖表以及和對應的 Rekognition 指標。如需詳細資訊，請參閱 CloudWatchRekognition 度量。

圖表	彙總指標
成功呼叫	SuccessfulRequestCount
客戶端錯誤	UserErrorCount
伺服器錯誤	ServerErrorCount

圖表	彙總指標
受節制	ThrottledCount
偵測到的標籤	DetectedLabelCount
偵測到的臉部	DetectedFaceCount

每個圖表顯示指定期間內收集到的彙總指標資料。也會顯示時間期間內彙總指標資料的總數。若要查看個別 API 呼叫的指標，請選擇各圖表下方的連結。

若要允許使用者存取 Rekognition 量度窗格，請確定使用者具有適當的 CloudWatch和 Rekognition 權限。例如，使用者若擁有 `AmazonRekognitionReadOnlyAccess` 和 `CloudWatchReadOnlyAccess` 受管政策的使用權限，便可查看指標窗格。如果使用者沒有所需的權限，開啟指標窗格時便不會顯示圖表。如需詳細資訊，請參閱 Amazon Rekognition 的身分與存取管理。

如需使用監視 Rekognition 的詳細資訊，請參閱。 CloudWatch 使用亞馬遜監控 Rekognition 要手段 CloudWatch

若要查看彙總指標 (主控台)

1. 前往 https://console.aws.amazon.com/rekognition/ 開啟 Amazon Rekognition 主控台。
2. 在導覽窗格中，選擇 指標。
3. 在下拉式清單中，選取您需要的指標所屬期間。
4. 若要更新圖表，請選擇 Refresh (重新整理) 按鈕。
5. 若要查看特定彙總 CloudWatch 量度的詳細度量，請選擇量度圖表 CloudWatch下方的「查看詳細資訊」。

使用映像和影片

您可以在三種不同類型的媒體上使用 Amazon Rekognition API 操作：影像、儲存的影片和串流影片。本節提供有關撰寫可存取 Amazon Rekognition 以處理不同類型媒體的程式碼的一般資訊。如需有關最佳做法和考量事項的指引，請參閱下面列出的各節，具體取決於您正在處理的媒體類型。

本指南中的其他章節提供特定映像和影片類型 (例如人臉偵測) 的分析相關資訊。

主題

- 使用映像
- 使用儲存的影片分析
- 使用串流視訊事件
- 錯誤處理
- 使用亞馬遜重新認知作為 FedRAMP 授權服務

使用映像

本節說明 Amazon Rekognition Image 可對映像執行的分析類型。

- 選擇物件和場景偵測
- 人臉偵測與比較
- 在集合中搜尋人臉
- 名人辨識
- 映像管制
- 映像偵測中的文字

這些分析皆由非儲存 API 操作執行，在這項操作中，Amazon Rekognition Image 不會儲存該操作探索到的任何資訊。非儲存 API 操作不會儲存任何輸入映像位元組。如需詳細資訊，請參閱 非儲存與以儲存為基礎的 API 操作。

Amazon Rekognition Image 也可將人臉中繼資料存放在集合中，以供稍後擷取。如需詳細資訊，請參閱 在集合中搜尋人臉。

在本節中，您會使用 Amazon Rekognition Image API 操作來分析存放在 Amazon S3 儲存貯體中的映像，以及從本機檔案系統載入的映像位元組。本節也說明如何從 .jpg 映像取得映像方向資訊。

Rekognition 只會使用 RGB 色版來執行推論。 AWS 建議使用者在使用顯示器之前先移除 Alpha 色版，以便直觀 (由人工手動) 檢查比較。

主題

映像規格

Amazon Rekognition Image 操作可以分析 .jpg 或 .png 格式的映像。

您可以將映像位元組做為呼叫的一部分傳遞至 Amazon Rekognition Image 操作，或參考現有的 Amazon S3 物件。如需分析存放在 Amazon S3 儲存貯體中映像的範例，請參閱 分析存放在 Amazon S3 儲存貯體中的映像。如需傳遞映像位元組到 Amazon Rekognition Image API 操作的範例，請參閱 分析從本機檔案系統載入的映像。

如果您使用 HTTP 並將映像位元組做為 Amazon Rekognition Image 操作的一部分來傳遞，映像位元組必須是 Base64 編碼字串。如果您使用 AWS SDK 並將映像位元組做為 API 操作呼叫的一部分來傳遞，則需視您使用的語言，對映像位元組進行 Base64 編碼。

下列常見的 AWS SDK 會自動對映像進行基礎 64 編碼，而且您不需要在呼叫 Amazon Rekognition 映像 API 作業之前對映像位元組進行編碼。

- Java
- JavaScript
- Python
- PHP

如果您使用其他 AWS SDK，並在呼叫 Rekognition API 操作時收到映像格式錯誤，請嘗試先對映像位元組進行 Base64 編碼，再傳遞至 Rekognition API 操作。

如果您使用 AWS CLI 呼叫 Amazon Rekognition 映像作業，則不支援在呼叫中傳遞映像位元組。您必須先將映像上傳至 Amazon S3 儲存貯體，再呼叫參考已上傳映像的操作。

> **Note**
>
> 如果您傳遞存放在 S3Object 中的映像而不是映像位元組，映像就不需經過 Base64 編碼。

如需有關確保 Amazon Rekognition Image 操作延遲盡可能降到最低的資訊，請參閱 Amazon Rekognition Image 操作延遲。

修正映像方向

在幾個 Rekognition API 操作中，會傳回已分析映像的方向。知道映像方向很重要，因為這可讓您改變映像的顯示方向。分析人臉的 Rekognition API 操作也會傳回映像中人臉位置的週框方塊。您可以使用週框方塊，在映像上的人臉周圍顯示方塊。傳回的週框方塊座標受映像方向所影響，所以您可能需要移動週框方塊座標，才能在人臉周圍正確地顯示方塊。如需詳細資訊，請參閱 取得映像方向與週框方塊座標。

映像大小調整

在分析期間，Amazon Rekognition 會使用一組最適合特定模型或演算法的預先定義範圍，在內部調整映像大小。因此，Amazon Rekognition 可能會偵測到不同數量的物件，或提供不同的結果，具體取決於輸入映像的解析度。例如，假設您有兩個映像。第一個映像具有 1024x768 像素的分辨率。第二個映像是第一個映像的重新調整大小的版本，解析度為 640x480 像素。如果您將影像送至 DetectLabels，兩次呼叫的回應DetectLabels可能會略有不同。

分析存放在 Amazon S3 儲存貯體中的映像

Amazon Rekognition Image 可以分析存放在 Amazon S3 儲存貯體中的映像，或做為映像位元組提供的映像。

在本主題中，您可以使用 DetectLabelsAPI 操作來偵測存放在 Amazon S3 儲存貯體的影像 (JPEG 或 PNG) 中的物件、概念和場景。您可以使用映像輸入參數，將映像傳遞至 Amazon Rekognition Image API 操作。在 Image 內，您指定 S3Object 物件屬性以參考存放在 S3 儲存貯體中的映像。存放在 Amazon S3 儲存貯體中的映像位元組，不需要 Base64 編碼。如需詳細資訊，請參閱 映像規格。

範例請求

在此範例中，JSON 要求 DetectLabels，而來源映像 (input.jpg) 是從名為 MyBucket 的 Amazon S3 儲存貯體載入。含有 S3 物件的 S3 儲存貯體區域必須符合您用於 Amazon Rekognition Image 操作的區域。

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

下列範例使用各種 AWS SDK 和呼叫DetectLabels。 AWS CLI 如需有關 DetectLabels 操作回應的資訊，請參閱 DetectLabels 回應。

偵測映像中的標籤

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。請確定您已為呼叫 API 操作的使用者授予程式設計存取的適當權限，請參閱 授與程式設計存取權 以取得如何執行此操作的指示。

2. 將包含一個或多個物件的映像 (例如樹、房子和船) 上傳至您的 S3 儲存貯體。映像的格式必須是 .jpg 或 .png 格式。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例來呼叫 DetectLabels 操作。

   Java

   此範例顯示一份在輸入映像中偵測到的標籤清單。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。

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

   package com.amazonaws.samples;
   ```

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.DetectLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.services.rekognition.model.S3Object;
import java.util.List;

public class DetectLabels {

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

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


      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      DetectLabelsRequest request = new DetectLabelsRequest()
           .withImage(new Image()
           .withS3Object(new S3Object()
           .withName(photo).withBucket(bucket)))
           .withMaxLabels(10)
           .withMinConfidence(75F);

      try {
         DetectLabelsResult result = rekognitionClient.detectLabels(request);
         List <Label> labels = result.getLabels();

         System.out.println("Detected labels for " + photo);
         for (Label label: labels) {
            System.out.println(label.getName() + ": " +
 label.getConfidence().toString());
         }
      } catch(AmazonRekognitionException e) {
         e.printStackTrace();
      }
   }
}
```

AWS CLI

此範例顯示 detect-labels CLI 操作的 JSON 輸出。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

如果您使用的是 Windows，則可能需要逸出引號，如以下範例所示。

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

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import java.util.List;

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
```

```
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectLabels {

    public static void main(String[] args) {

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

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

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

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

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    public static void getLabelsfromImage(RekognitionClient rekClient, String
 bucket, String image) {

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

```
                .build() ;

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

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

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

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
 // snippet-end:[rekognition.java2.detect_labels.main]
}
```

Python

此範例顯示在輸入映像中偵測到的標籤。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3

def detect_labels(photo, bucket):
```

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

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

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

        for instance in label['Instances']:
            print(" Bounding box")
            print(" Top: " + str(instance['BoundingBox']['Top']))
            print(" Left: " + str(instance['BoundingBox']['Left']))
            print(" Width: " + str(instance['BoundingBox']['Width']))
            print(" Height: " + str(instance['BoundingBox']['Height']))
            print(" Confidence: " + str(instance['Confidence']))
            print()

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

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

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

    if "ImageProperties" in str(response):
        print("Background:")
```

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

        return len(response['Labels'])

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

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

Node.Js

此範例顯示與映像中偵測到的名人有關的資訊。

將 photo 的值變更為某個映像檔案的路徑和檔案名稱，而該映像檔案含有一個或多個名人臉孔。將 bucket 的值變更為包含映像檔案的 S3 儲存貯體名稱。將 REGION 的值變更為與您帳戶相關聯的地區名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

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

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

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

```
  }),
});

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

// Set params
const params = {For example, to grant
    Image: {
      S3Object: {
        Bucket: bucket,
        Name: photo
      },
    },
  }

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

detect_labels();
```

.NET

此範例顯示一份在輸入映像中偵測到的標籤清單。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

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

        try
        {
            DetectLabelsResponse detectLabelsResponse =
 rekognitionClient.DetectLabels(detectlabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (Label label in detectLabelsResponse.Labels)
```

```
                Console.WriteLine("{0}: {1}", label.Name, label.Confidence);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

Ruby

此範例顯示一份在輸入映像中偵測到的標籤清單。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。

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

```
      puts "  Bounding box:"
      puts "    Top:        #{box.top}"
      puts "    Left:       #{box.left}"
      puts "    Width:      #{box.width}"
      puts "    Height:     #{box.height}"
      puts "  Confidence: #{instance.confidence}"
    end
    puts "Parents:"
    label.parents.each do |parent|
      puts "  #{parent.name}"
    end
    puts "------------"
    puts ""
  end
```

回應範例

DetectLabels 的回應是映像中偵測到的一系列標籤，以及偵測所依據的可信度層級。

當您對映像執行 DetectLabels 作業時，Amazon Rekognition 會傳回類似下列範例回應的輸出。

回應顯示操作偵測到多個標籤，包括人員、車輛和汽車。每個標籤都有一個相關的可信度等級。例如，偵測演算法對於映像中包含人員的可信度為 98.991432%。

回應也包含 Parents 陣列中標籤的上階標籤。例如 Automobile (汽車) 標籤有兩個名為 Vehicle (車輛) 和 Transportation (運輸) 的 父標籤。

常見物件標籤的回應包含輸入映像上標籤位置的週框方塊資訊。例如，人員標籤具有一個實例陣列，其中包含兩個週框方塊。這些是在映像中偵測到的兩個人員位置。

欄位 LabelModelVersion 包含 DetectLabels 所使用之偵測模型的版本編號。

如需使用此 DetectLabels 操作的詳細資訊，請參閱 偵測物件和概念。

```
{

    {
    "Labels": [
        {
            "Name": "Vehicle",
            "Confidence": 99.15271759033203,
            "Instances": [],
```

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

```
                    },
                    "Confidence": 99.1286392211914
                },
                {
                    "BoundingBox": {
                        "Width": 0.14233611524105072,
                        "Height": 0.15528248250484467,
                        "Left": 0.6494812965393066,
                        "Top": 0.5333095788955688
                    },
                    "Confidence": 98.48368072509766
                },
                {
                    "BoundingBox": {
                        "Width": 0.11086395382881165,
                        "Height": 0.10271988064050674,
                        "Left": 0.10355594009160995,
                        "Top": 0.5354844927787781
                    },
                    "Confidence": 96.45606231689453
                },
                {
                    "BoundingBox": {
                        "Width": 0.06254628300666809,
                        "Height": 0.053911514580249786,
                        "Left": 0.46083059906959534,
                        "Top": 0.5573825240135193
                    },
                    "Confidence": 93.65448760986328
                },
                {
                    "BoundingBox": {
                        "Width": 0.10105438530445099,
                        "Height": 0.12226245552301407,
                        "Left": 0.5743985772132874,
                        "Top": 0.534368634223938
                    },
                    "Confidence": 93.06217193603516
                },
                {
                    "BoundingBox": {
                        "Width": 0.056389667093753815,
                        "Height": 0.17163699865341187,
                        "Left": 0.9427769780158997,
```

```
                    "Top": 0.5235804319381714
                },
                "Confidence": 92.6864013671875
            },
            {
                "BoundingBox": {
                    "Width": 0.06003860384225845,
                    "Height": 0.06737709045410156,
                    "Left": 0.22409997880458832,
                    "Top": 0.5441341400146484
                },
                "Confidence": 90.4227066040039
            },
            {
                "BoundingBox": {
                    "Width": 0.02848697081208229,
                    "Height": 0.19150497019290924,
                    "Left": 0.0,
                    "Top": 0.5107086896896362
                },
                "Confidence": 86.65286254882812
            },
            {
                "BoundingBox": {
                    "Width": 0.04067881405353546,
                    "Height": 0.03428703173995018,
                    "Left": 0.316415935754776,
                    "Top": 0.5566273927688599
                },
                "Confidence": 85.36471557617188
            },
            {
                "BoundingBox": {
                    "Width": 0.043411049991846085,
                    "Height": 0.0893595889210701,
                    "Left": 0.18293385207653046,
                    "Top": 0.5394920110702515
                },
                "Confidence": 82.21705627441406
            },
            {
                "BoundingBox": {
                    "Width": 0.031183116137981415,
                    "Height": 0.03989990055561066,
```

```
                    "Left": 0.2853088080883026,
                    "Top": 0.5579366683959961
                },
                "Confidence": 81.0157470703125
            },
            {
                "BoundingBox": {
                    "Width": 0.031113790348172188,
                    "Height": 0.056484755128622055,
                    "Left": 0.2580395042896271,
                    "Top": 0.5504819750785828
                },
                "Confidence": 56.13441467285156
            },
            {
                "BoundingBox": {
                    "Width": 0.08586374670267105,
                    "Height": 0.08550430089235306,
                    "Left": 0.5128012895584106,
                    "Top": 0.5438792705535889
                },
                "Confidence": 52.37760925292969
            }
        ],
        "Parents": [
            {
                "Name": "Vehicle"
            },
            {
                "Name": "Transportation"
            }
        ]
    },
    {
        "Name": "Human",
        "Confidence": 98.9914321899414,
        "Instances": [],
        "Parents": []
    },
    {
        "Name": "Person",
        "Confidence": 98.9914321899414,
        "Instances": [
            {
```

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


}
```

分析從本機檔案系統載入的映像

Amazon Rekognition Image 操作可以分析做為映像位元組提供的映像，或存放在 Amazon S3 儲存貯體中的映像。

這些主題提供範例，其示範透過使用從本機檔案系統載入的檔案，將映像位元組提供給 Amazon Rekognition Image API 操作。您可以使用映像輸入參數，將映像位元組傳遞至 Amazon Rekognition API 操作。在 `Image` 內，指定 `Bytes` 屬性，以傳遞 base64 編碼的映像位元組。

透過使用 `Bytes` 輸入參數傳遞至 Amazon Rekognition API 操作的映像，必須經過 Base64 編碼。這些範例使用的 AWS SDK，會自動以 base64 編碼映像。在呼叫 Amazon Rekognition API 操作前，您不需要再編碼映像位元組。如需詳細資訊，請參閱 映像規格。

在這個範例中，JSON 請求 `DetectLabels`，來源映像位元組以 `Bytes` 輸入參數傳遞。

```
{
```

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

下列範例使用各種 AWS SDK 和呼叫DetectLabels。 AWS CLI 如需有關 DetectLabels 操作回應的資訊，請參閱 DetectLabels 回應。

如需用戶端 JavaScript 範例，請參閱使用 JavaScript。

偵測本機映像中的標籤

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 DetectLabels 操作。

   Java

   下列 Java 範例示範如何從本機檔案系統載入映像，並運用 detectLabels AWS SDK 操作來偵測標籤。將 photo 的值變更為某個映像檔案的路徑和檔案名稱 (.jpg 或 .png 格式)。

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

package aws.example.rekognition.image;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.AmazonClientException;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.DetectLabelsRequest;
```

```
import com.amazonaws.services.rekognition.model.DetectLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.util.IOUtils;

public class DetectLabelsLocalFile {
    public static void main(String[] args) throws Exception {
     String photo="input.jpg";


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


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        DetectLabelsRequest request = new DetectLabelsRequest()
                .withImage(new Image()
                        .withBytes(imageBytes))
                .withMaxLabels(10)
                .withMinConfidence(77F);

        try {

            DetectLabelsResult result =
 rekognitionClient.detectLabels(request);
            List <Label> labels = result.getLabels();

            System.out.println("Detected labels for " + photo);
            for (Label label: labels) {
               System.out.println(label.getName() + ": " +
 label.getConfidence().toString());
            }

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

    }
}
```

Python

下列適用於 Python 的 AWS SDK 範例示範如何從本機檔案系統載入映像，並呼叫 detect_labels 操作。將 photo 的值變更為某個映像檔案的路徑和檔案名稱 (.jpg 或 .png 格式)。

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

import boto3

def detect_labels_local_file(photo):


    client=boto3.client('rekognition')

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

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

    return len(response['Labels'])

def main():
    photo='photo'

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


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

.NET

下列 Java 範例示範如何從本機檔案系統載入映像，並運用 DetectLabels 操作來偵測標籤。將 photo 的值變更為某個映像檔案的路徑和檔案名稱 (.jpg 或 .png 格式)。

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

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
```

```
        {
            Image = image,
            MaxLabels = 10,
            MinConfidence = 77F
        };

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

PHP

以下 AWS SDK for PHP 件範例說明如何從本機檔案系統載入映像並呼叫 DetectFacesAPI 作業。將 photo 的值變更為某個映像檔案的路徑和檔案名稱 (.jpg 或 .png 格式)。

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

    require 'vendor/autoload.php';

    use Aws\Rekognition\RekognitionClient;

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

    $rekognition = new RekognitionClient($options);
```

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


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

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

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

Ruby

此範例顯示一份在輸入映像中偵測到的標籤清單。將 photo 的值變更為某個映像檔案的路徑和檔案名稱 (.jpg 或 .png 格式)。

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

    # gem 'aws-sdk-rekognition'
    require 'aws-sdk-rekognition'
    credentials = Aws::Credentials.new(
```

```
       ENV['AWS_ACCESS_KEY_ID'],
       ENV['AWS_SECRET_ACCESS_KEY']
   )
   client   = Aws::Rekognition::Client.new credentials: credentials
   photo = 'photo.jpg'
   path = File.expand_path(photo) # expand path relative to the current
directory
   file = File.read(path)
   attrs = {
     image: {
       bytes: file
     },
     max_labels: 10
   }
   response = client.detect_labels attrs
   puts "Detected labels for: #{photo}"
   response.labels.each do |label|
     puts "Label:      #{label.name}"
     puts "Confidence: #{label.confidence}"
     puts "Instances:"
     label['instances'].each do |instance|
       box = instance['bounding_box']
       puts "  Bounding box:"
       puts "    Top:        #{box.top}"
       puts "    Left:       #{box.left}"
       puts "    Width:      #{box.width}"
       puts "    Height:     #{box.height}"
       puts "  Confidence: #{instance.confidence}"
     end
     puts "Parents:"
     label.parents.each do |parent|
       puts "  #{parent.name}"
     end
     puts "------------"
     puts ""
   end
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

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

                Usage:    <sourceImage>

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

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

        String sourceImage = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectImageLabels(rekClient, sourceImage);
        rekClient.close();
```

```
    }

    public static void detectImageLabels(RekognitionClient rekClient, String
 sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

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

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

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

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

使用 JavaScript

以下 JavaScript 網頁示例允許用戶選擇圖像並查看在圖像中檢測到的臉部的估計年齡。預估的年齡是通過電話返回DetectFaces。

所選擇的圖像通過使用該 JavaScript`FileReader.readAsDataURL`功能加載，該功能 base64 對圖像進行編碼。此步驟對於顯示 HTML 畫布上的映像有幫助。但是，這表示映像位元組需要在傳遞到

Amazon Rekognition Image 操作前解碼。此範例說明如何解碼已載入的映像位元組。如果已編碼的映像位元組對您來說沒有幫助，請改用 `FileReader.readAsArrayBuffer`，因為載入的映像未經過編碼。這表示可以在未先解碼映像位元組的情況下呼叫 Amazon Rekognition Image 操作。如需範例，請參閱使用 readAsArray緩衝區。

若要執行 JavaScript 範例

1. 載入範例原始程式碼到編輯器。
2. 取得 Amazon Cognito 身分池識別碼。如需詳細資訊，請參閱 獲取 Amazon Cognito 身分池識別碼。
3. 在範本程式碼的 `AnonLog` 函數中，變更 `IdentityPoolIdToUse` 與 `RegionToUse` 為您在 獲取 Amazon Cognito 身分池識別碼 步驟 9 中記下的值。
4. 在 `DetectFaces` 函數中，變更 `RegionToUse` 為您在之前的步驟中使用的值。
5. 將範例原始程式碼儲存為 `.html` 檔案。
6. 載入檔案到您的瀏覽器。
7. 選擇瀏覽... 按鈕，然後選擇其中包含一或多個人臉的映像。將顯示一個資料表，其中包含每個於映像中偵測到的人臉之估測年齡。

> **Note**
>
> 以下程式碼範例使用不再是 Amazon Cognito 一部分的兩個指令碼。要獲取這些文件，請點擊 .min.js 和 aws-cognito-sdk.min.js 的鏈接，然後將每個文件中的文本保存為單獨的文件 amazon-cognito-identity。 `.js`

JavaScript 範例程式碼

下列程式碼範例使用 JavaScript V2。如需 JavaScript V3 中的範例，請參閱AWS 文件 SDK 範例 GitHub 儲存庫中的範例。

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

```
  <script src="aws-cognito-sdk.min.js"></script>
  <script src="amazon-cognito-identity.min.js"></script>
  <script src="https://sdk.amazonaws.com/js/aws-sdk-2.16.0.min.js"></script>
  <meta charset="UTF-8">
  <title>Rekognition</title>
</head>

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

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

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

```
//Loads selected image and unencodes image bytes for Rekognition DetectFaces API
function ProcessImage() {
  AnonLog();
  var control = document.getElementById("fileToUpload");
  var file = control.files[0];

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

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

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

使用 readAsArray緩衝區

下列程式碼片段是使用 JavaScript V2 範例程式碼中`ProcessImage`函式的替代實作。它使用 `readAsArrayBuffer` 來載入映像並呼叫 `DetectFaces`。由於 `readAsArrayBuffer` 不會將載入的檔案以 Base64 編碼，在呼叫 Amazon Rekognition Image 操作前無需解碼映像位元組。

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

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

    // Load base64 encoded image for display
    var reader = new FileReader();
    reader.onload = (function (theFile) {
      return function (e) {
        //Call Rekognition
        AWS.region = "RegionToUse";
        var rekognition = new AWS.Rekognition();
        var params = {
          Image: {
          Bytes: e.target.result
        },
        Attributes: [
```

```
        'ALL',
      ]
    };
    rekognition.detectFaces(params, function (err, data) {
      if (err) console.log(err, err.stack); // an error occurred
      else {
       var table = "<table><tr><th>Low</th><th>High</th></tr>";
        // show each face and build out estimated age table
        for (var i = 0; i < data.FaceDetails.length; i++) {
          table += '<tr><td>' + data.FaceDetails[i].AgeRange.Low +
            '</td><td>' + data.FaceDetails[i].AgeRange.High + '</td></tr>';
        }
        table += "</table>";
        document.getElementById("opResult").innerHTML = table;
      }
    });

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

獲取 Amazon Cognito 身分池識別碼

為簡化程序，範例將使用匿名 Amazon Cognito 身分池來提供 Amazon Rekognition Image API 未經授權的存取。這可能符合您的需求。例如，您可以使用未經驗證的存取來提供使用者在註冊前的免費存取或試用存取您的網站。為了提供經驗證的存取，使用 Amazon Cognito 使用者集區。如需詳細資訊，請參閱 Amazon Cognito 使用者集區。

下列程序說明如何建立身分池，以允許存取未經驗證的身分，以及如何取得在範本程式碼中所需的身分池識別碼。

若要取得身分池識別碼

1. 開啟 Amazon Cognito 主控台。
2. 選擇 Create new identity pool (建立新的身分池)。
3. 在 Identity pool name* (身分池名稱*) 中，輸入您的身分池名稱。
4. 在 Unauthenticated identites (未經驗證的身分) 中，選擇 Enable access to unauthenticated identities (允許存取未經驗證的身分) 。
5. 選擇 Create Pool (建立集區)。

6. 選擇 View Details (檢視詳細資訊)，並記下未經驗證的身分之角色名稱。
7. 選擇 Allow (允許)。
8. 在「平台」中，選擇JavaScript。
9. 在 Get AWS Credentials (取得 AWS 憑證) 中，請記下程式碼片段中顯示的 `AWS.config.region` 和 `IdentityPooldId` 值。
10. 在以下網址開啟 IAM 主控台：https://console.aws.amazon.com/iam/。
11. 在導覽窗格中，選擇角色。
12. 選擇您在步驟 6 記下的角色名稱。
13. 在 Permissions (許可) 索引標籤中，選擇 Attach policies (連接政策)。
14. 選擇 [AmazonRekognitionReadOnly存取]。
15. 選擇 Attach Policy (連接政策)。

顯示週框方塊

Amazon Rekognition Image 操作可以針對映像中偵測到的專案傳回週框方塊。例如，此DetectFaces作業會針對影像中偵測到的每個臉部傳回邊界方框 (BoundingBox)。您可以使用週框方塊座標，來顯示所偵測到之專案的周圍方塊。例如，以下映像會顯示圍繞人臉的週框方塊。

BoundingBox 具有下列屬性：

- 高度：週框方塊的高度，以整體映像高度的比例表示。
- 左邊：週框方塊的左邊座標，以整體映像寬度的比例表示。
- 上方：週框方塊的上方座標，以整體映像高度的比例表示。
- 寬度：週框方塊的寬度，以整體映像寬度的比例表示。

每個 BoundingBox 屬性都有一個介於 0 和 1 之間的值。每個屬性值是整體映像寬度 (Left 和 Width) 或高度 (Height 和 Top) 的比例。例如，如果輸入映像為 700 x 200 像素，而週框方塊的左上方座標為 350 x 50 像素，則 API 會傳回 Left 值 0.5 (350/700) 和 Top 值 0.25 (50/200)。

下圖顯示每個週框方塊屬性涵蓋的映像範圍。

要顯示具有正確位置和大小的邊界框，您必須將 BoundingBox 值乘以圖像寬度或高度（取決於您想要的值）才能獲得像素值。您可以使用像素值來顯示週框方塊。例如，前一個映像的像素維度為 608 寬度 x 588 高度。臉部的週框方塊值如下：

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

臉部週框方塊的位置 (以像素表示) 的計算方式如下：

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

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

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

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

您可以使用這些值來顯示臉部的週框方塊。

> **Note**
>
> 您可以透過多種方式導向映像。您的應用程式可能需要旋轉映像，才能以正確方向顯示它。週框方塊座標會受到映像方向的影響。您可能需要轉換座標，然後才能在正確位置顯示週框方塊。如需詳細資訊，請參閱 取得映像方向與週框方塊座標。

下列範例顯示如何在透過呼叫偵測到的面周圍顯示邊界方塊DetectFaces。這些範例假設映像導向至 0 度。這些範例也會顯示如何從 Amazon S3 儲存貯體下載映像。

顯示週框方塊

1. 如果您尚未執行：

a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 DetectFaces 操作。

Java

將 bucket 的值變更為包含映像檔案的 Amazon S3 儲存貯體。將 photo 的值變更為某個映像檔案的檔案名稱 (.jpg 或 .png 格式)。

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

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

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

import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.DetectFacesRequest;
import com.amazonaws.services.rekognition.model.DetectFacesResult;
import com.amazonaws.services.rekognition.model.FaceDetail;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

// Calls DetectFaces and displays a bounding box around each detected image.
public class DisplayFaces extends JPanel {

    private static final long serialVersionUID = 1L;
```

```
   BufferedImage image;
   static int scale;
   DetectFacesResult result;

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

       result = facesResult;
       image = bufImage;


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

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

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

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

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

       }
   }


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

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

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

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

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

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

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

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

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

        }

        // Create frame and panel.
```

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

    }
}
```

Python

將 bucket 的值變更為包含映像檔案的 Amazon S3 儲存貯體。將 photo 的值變更為某個映像檔案的檔案名稱 (.jpg 或 .png 格式)。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

def show_faces(photo, bucket):

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

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

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

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

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

```
    # calculate and display bounding boxes for each detected face
    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']
['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')

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

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

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

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

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

    image.show()

    return len(response['FaceDetails'])

def main():
    bucket = "bucket-name"
    photo = "photo-name"
    faces_count = show_faces(photo, bucket)
    print("faces detected: " + str(faces_count))

if __name__ == "__main__":
```

```
    main()
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

請注意，s3 是指 AWS SDK Amazon Amazon S3 客戶端，并且 rekClient 是指 AWS SDK Amazon Rekognition 客戶端。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
//snippet-end:[rekognition.java2.detect_labels.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
```

```
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DisplayFaces extends JPanel {

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

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

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

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

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

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

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

```
   // snippet-start:[rekognition.java2.display_faces.main]
   public static void displayAllFaces(S3Client s3,
                                      RekognitionClient rekClient,
                                      String sourceImage,
                                      String bucketName) {
       int height;
       int width;
       byte[] data = getObjectBytes (s3, bucketName, sourceImage);
       InputStream is = new ByteArrayInputStream(data);

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

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

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

           result = rekClient.detectFaces(facesRequest);

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

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

```
                System.out.println();
            }

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

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

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

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

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

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

    public DisplayFaces(BufferedImage bufImage) {
        super();
```

```
            scale = 1; // increase to shrink image size.
            image = bufImage;
        }

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

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

            // Iterate through the faces and display bounding boxes.
            List<FaceDetail> faceDetails = result.faceDetails();
            for (FaceDetail face : faceDetails) {
                BoundingBox box = face.boundingBox();
                left = width * box.left();
                top = height * box.top();
                g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                        Math.round((width * box.width()) / scale),
     Math.round((height * box.height())) / scale);
            }
        }
        // snippet-end:[rekognition.java2.display_faces.main]
    }
```

取得映像方向與週框方塊座標

使用 Amazon Rekognition Image 的應用程式通常需要顯示 Amazon Rekognition Image 操作偵測到的映像，並在偵測到的人臉周圍顯示方塊。若要在您的應用程式中正確顯示映像，您需要了解映像的方向。您可能需要更正此方向。對於某些 .jpg 檔案，映像的方向會包含在映像的可交換映像檔案格式 (Exif) 中繼資料內。

若要在臉部周圍顯示方塊，您需要臉部週框方塊的座標。如果方塊的方向不正確，您可能需要調整這些座標。Amazon Rekognition Image 人臉偵測操作會針對每個偵測到的人臉傳回邊界框座標，但不會估計沒有 Exif 中繼資料的 .jpg 檔案的座標。

下列範例顯示如何獲得映像中偵測到之人臉的週框方塊座標。

請使用此範例中的資訊來確保您的映像方向正確，以及確保週框方塊在您應用程式中顯示的位置正確。

由於用來旋轉及顯示映像與週框方塊的程式碼取決於您所使用的語言和環境，因此我們不會說明如何在您的程式碼中顯示映像與週框方塊，也不會說明如何從 Exif 中繼資料取得方向資訊。

找出映像的方向

若要在您的應用程式中正確顯示映像，您可能需要加以旋轉。下列映像旋轉為 0 度並正確顯示。

不過，下列映像卻逆時針旋轉 90 度。若要正確顯示，您需要找出映像的方向，並在您的程式碼中使用該資訊將映像旋轉為 0 度。

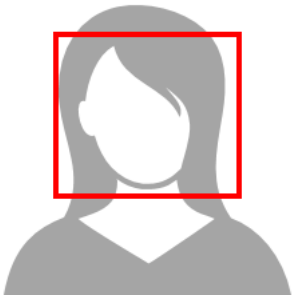
某些 .jpg 格式的映像在 Exif 中繼資料內含有方向資訊。如果可用，映像的 Exif 中繼資料會包含方向。在 Exif 中繼資料內，您可以在 `orientation` 欄位中找到映像的方向。雖然 Amazon Rekognition Image 會識別 Exif 中繼資料內具有映像方向資訊，但並不提供該資訊的存取權。若要存取映像中的 Exif 中繼資料，請使用第三方程式庫或自行撰寫程式碼。如需詳細資訊，請參閱可交換映像檔案格式版本 2.32。

當您知道映像的方向後，就可以撰寫程式碼加以旋轉並正確顯示。

顯示週框方塊

分析映像中人臉的 Amazon Rekognition Image 操作也會傳回人臉周圍的週框方塊座標。如需詳細資訊，請參閱BoundingBox。

若要在您的應用程式中於人臉周圍顯示週框方塊 (類似下圖所示的方塊)，請在您的程式碼中使用週框方塊座標。操作傳回的週框方塊座標會反映映像的方向。如果您必須旋轉映像才能將其正確顯示，可能需要移動週框方塊座標。



在 Exif 中繼資料內有方向資訊時顯示週框方塊

如果 Exif 中繼資料包含映像的方向，Amazon Rekognition Image 操作會執行下列操作：

- 在操作回應的方向修正欄位中傳回 null。若要旋轉映像，請在您的程式碼中使用 Exif 中繼資料提供的方向。
- 傳回已旋轉為 0 度的週框方塊座標。若要在正確位置顯示週框方塊，請使用傳回的座標。您不需要移動該座標。

範例：取得映像方向與映像的週框方塊座標

下列範例示範如何使用適用於 AWS SDK 來取得 Exif 映像的方向資料，並透過 `RecognizeCelebrities` 操作偵測到之名人的週框方塊座標。

> **Note**
>
> 自 2021 年 8 月起，已停止使用該 `OrientationCorrection` 欄位估算映像方向的支援。API 回應中包含之此欄位的任何傳回值將永遠為零。

Java

此範例會從本機檔案系統載入映像、呼叫 `RecognizeCelebrities` 操作、判斷映像的高度與寬度，並計算旋轉映像的人臉週框方塊座標。此範例不會示範如何處理存放在 Exif 中繼資料內的方向資訊。

在 main 函數中，請將 photo 的值取代為以 .png 或 .jpg 格式存放於本機之映像的名稱與路徑。

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

package com.amazonaws.samples;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.util.List;
import javax.imageio.ImageIO;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesRequest;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesResult;
import com.amazonaws.util.IOUtils;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.Celebrity;
import com.amazonaws.services.rekognition.model.ComparedFace;

public class RotateImage {

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

  String photo = "photo.png";

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


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

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

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

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

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

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

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

 //Call GetCelebrities

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


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

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

```
                    width,
                    face.getBoundingBox(),
                    result.getOrientationCorrection());

            System.out.println();
        }

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

}


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

  float left = 0;
  float top = 0;

  if(rotation==null){
      System.out.println("No estimated estimated orientation. Check Exif data.");
      return;
  }
  //Calculate face position based on image orientation.
  switch (rotation) {
     case "ROTATE_0":
        left = imageWidth * box.getLeft();
        top = imageHeight * box.getTop();
        break;
     case "ROTATE_90":
        left = imageHeight * (1 - (box.getTop() + box.getHeight()));
        top = imageWidth * box.getLeft();
        break;
     case "ROTATE_180":
        left = imageWidth - (imageWidth * (box.getLeft() + box.getWidth()));
        top = imageHeight * (1 - (box.getTop() + box.getHeight()));
        break;
     case "ROTATE_270":
        left = imageHeight * box.getTop();
        top = imageWidth * (1 - box.getLeft() - box.getWidth());
        break;
     default:
```

```
        System.out.println("No estimated orientation information. Check Exif
 data.");
        return;
  }

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

  }
}
```

Python

此範例使用 PIL/枕狀映像程式庫來取得映像寬度和高度。如需詳細資訊，請參閱枕狀。此範例會保留 exif 中繼資料，您可能需要應用程式中的其他位置。

在 main 函數中，請將 photo 的值取代為以 .png 或 .jpg 格式存放於本機之映像的名稱與路徑。

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

import boto3
import io
from PIL import Image


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

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

```
def celebrity_image_information(photo):
    client = boto3.client('rekognition')

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

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

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

    response = client.recognize_celebrities(Image={'Bytes': image_binary})

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

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

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

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


def main():
    photo = 'photo'

    celebrity_count = celebrity_image_information(photo)
```

```
    print("celebrities detected: " + str(celebrity_count))


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

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

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

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

                Usage:    <sourceImage>

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

```
                """;

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

        String sourceImage = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

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

    public static void recognizeAllCelebrities(RekognitionClient rekClient, String
sourceImage) {
        try {
            BufferedImage image;
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

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

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

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

            RecognizeCelebritiesResponse result =
rekClient.recognizeCelebrities(request);
            List<Celebrity> celebs = result.celebrityFaces();
            System.out.println(celebs.size() + " celebrity(s) were recognized.\n");
            for (Celebrity celebrity : celebs) {
                System.out.println("Celebrity recognized: " + celebrity.name());
```

```
                System.out.println("Celebrity ID: " + celebrity.id());
                ComparedFace face = celebrity.face();
                ShowBoundingBoxPositions(height,
                        width,
                        face.boundingBox(),
                        result.orientationCorrectionAsString());
            }

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

    public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
BoundingBox box, String rotation) {
        float left;
        float top;
        if (rotation == null) {
            System.out.println("No estimated estimated orientation.");
            return;
        }

        // Calculate face position based on the image orientation.
        switch (rotation) {
            case "ROTATE_0" -> {
                left = imageWidth * box.left();
                top = imageHeight * box.top();
            }
            case "ROTATE_90" -> {
                left = imageHeight * (1 - (box.top() + box.height()));
                top = imageWidth * box.left();
            }
            case "ROTATE_180" -> {
                left = imageWidth - (imageWidth * (box.left() + box.width()));
                top = imageHeight * (1 - (box.top() + box.height()));
            }
            case "ROTATE_270" -> {
                left = imageHeight * box.top();
                top = imageWidth * (1 - box.left() - box.width());
            }
            default -> {
```

```
                System.out.println("No estimated orientation information. Check Exif
 data.");
                return;
            }
        }

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

使用儲存的影片分析

Amazon Rekognition Video 為可用於分析影片的 API。使用 Amazon Rekognition Video，您可以偵測儲存於 Amazon Simple Storage Service (Amazon S3) 儲存貯體的影片中之標籤、人臉、人物、名人以及成人內容 (包括暗示與露骨)。您可以在媒體/娛樂與公共安全等類別使用 Amazon Rekognition Video。在過去，掃描影片中的物件或人員可能必須耗費數小時的人工檢視，且容易發生錯誤。若專案貫穿影片各處，Amazon Rekognition Video 可對專案執行自動化偵測。

本節涵蓋 Amazon Rekognition Video 可執行的分析類型、API 概觀以及使用 Amazon Rekognition Video 的範例。

主題

- 分析類型
- Amazon Rekognition Video API 概觀
- 呼叫 Amazon Rekognition Video 操作
- 設定 Amazon Rekognition Video
- 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片
- 使用分析視訊 AWS Command Line Interface
- 參考：影片分析結果通知
- 對 Amazon Rekognition Video 進行疑難排解

分析類型

您可以使用 Amazon Rekognition Video 來分析影片中的下列資訊：

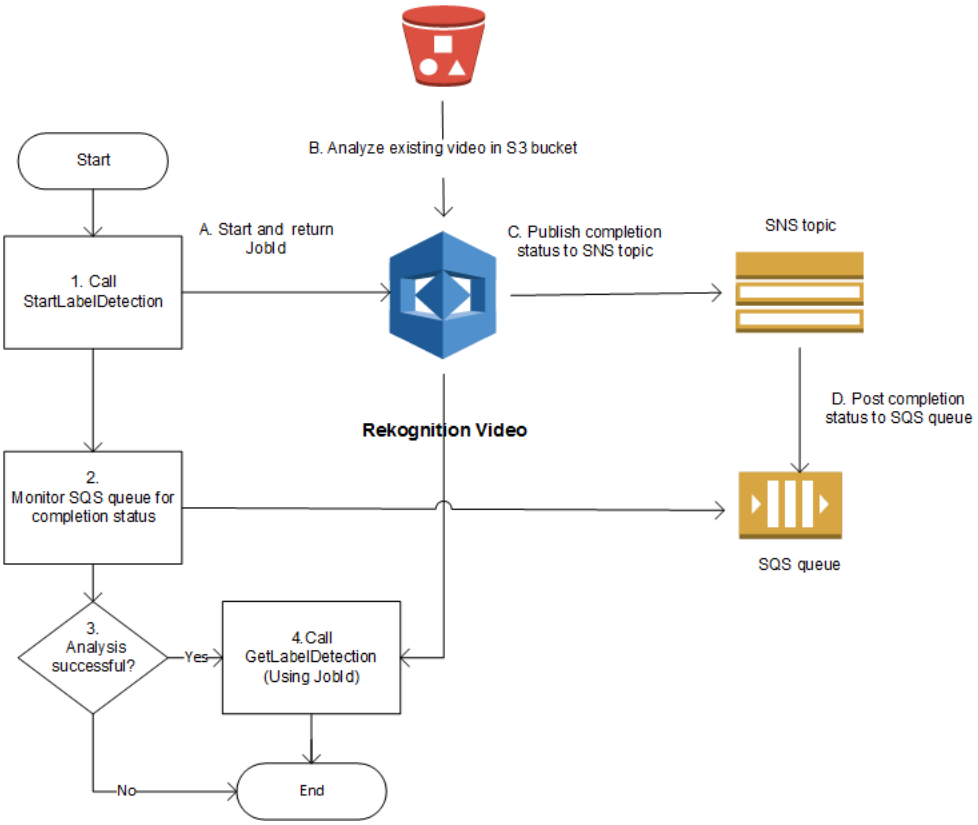
- 影片區段
- 標籤
- 暗示與露骨的成人內容
- Text (文字)
- 名人
- 臉部
- 人物

如需詳細資訊，請參閱 Amazon Rekognition 的運作方式。

Amazon Rekognition Video API 概觀

Amazon Rekognition Video 會處理存放在 Amazon S3 儲存貯體中的影片。設計模式為一組非同步的操作。您可以透過呼叫諸如此類的`Start`作業來開始視訊分析StartLabelDetection。請求的完成狀態會發佈至 Amazon Simple Notification Service (Amazon SNS) 主題。若要從 Amazon SNS 主題取得完成狀態，您可以使用 Amazon Simple Queue Service (Amazon SQS) 佇列或 AWS Lambda 函數。取得完成狀態之後，您可以呼叫`Get`作業，例如 GetLabelDetection，以取得要求的結果。

下圖顯示在儲存於 Amazon S3 儲存貯體的影片中偵測標籤的程序。在圖表內，佇列將自 Amazon SNS 主題取得完成狀態。或者，您可以使用 AWS Lambda 函數。



其他 Amazon Rekognition Video 操作的過程是相同的。下表列出每個非儲存體 Amazon Rekognition 操作的 `Start` 與 `Get` 操作。

偵測	開始操作	取得操作
影片區段	StartSegmentDetection	GetSegmentDetection
標籤	StartLabelDetection	GetLabelDetection
露骨或暗示性的成人內容	StartContentModeration	GetContentModeration
文字	StartTextDetection	GetTextDetection
名人	StartCelebrityRecognition	GetCelebrityRecognition
臉部	StartFaceDetection	GetFaceDetection
人員	StartPersonTracking	GetPersonTracking

對於 Get 以外的 GetCelebrityRecognition 操作，Amazon Rekognition Video 將傳回追蹤資訊，說明將在影片輸入檔的哪個部分偵測到實體。

如需使用 Amazon Rekognition Video 影片的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。如需使用 Amazon SQS 執行影片分析的範例，請參閱 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。如需 AWS CLI 範例，請參閱使用分析視訊 AWS Command Line Interface。

影片格式和儲存體

Amazon Rekognition 操作可分析儲存於 Amazon S3 儲存貯體的影片。如需影片分析作業的所有限制清單，請參閱 指導方針和配額。

影片需使用 H.264 編解碼器來編碼。支援的檔案格式為 MPEG-4 與 MOV。

編解碼器為壓縮資料讓傳遞更快速並將收到的資料解壓縮為原始格式的軟體或硬體工具。The H.264 編解碼器通常用於記錄、壓縮和發佈影片內容。影片檔案格式可包含一個或多個編解碼器。若您的 MOV 或 MPEG-4 格式影片檔案不適用於 Amazon Rekognition Video，請確認用於影片編碼的編解碼器為 H.264。

任何可分析音訊資料的 Amazon Rekognition Video API 只支援 AAC 音訊轉碼器。

可儲存影片的檔案大小上限為 10 GB。

搜尋人物

您可以使用儲存在集合中的臉部中繼資料來搜尋影片中的人物。例如，您可以在封存的影片中搜尋特定人物或多個人物。您可以使用IndexFaces操作將來源影像的臉部中繼資料儲存在集合中。然後，您可以使用StartFaceSearch以非同步方式開始搜尋集合中的臉孔。您可GetFaceSearch以使用來取得搜尋結果。如需詳細資訊，請參閱 在儲存的影片中搜尋人臉。搜尋人物為以儲存體為基礎的 Amazon Rekognition 操作之範例。如需詳細資訊，請參閱 以儲存為基礎的 API 操作。

您也可以在串流影片中搜尋人物。如需詳細資訊，請參閱 使用串流視訊事件。

呼叫 Amazon Rekognition Video 操作

Amazon Rekognition Video 是一種非同步 API，可以用來分析存放在 Amazon Simple Storage Service (Amazon S3) 儲存貯體中的影片。您可以透過呼叫亞馬遜 Rekognition 視訊操作，例如開始對視訊進Start行分析。StartPersonTrackingAmazon Rekognition Video 會將分析請求的結果發佈至 Amazon Simple Notification Service (Amazon SNS) 主題。您可以使用 Amazon Simple Queue Service (Amazon SQS) 佇列或 AWS Lambda 函數，從 Amazon SNS 主題取得影片分析請

求的完成狀態。最後，您可以透過呼叫 Amazon Rekognition Get 作業取得視訊分析請求結果，例如。GetPersonTracking

下節中的資訊使用標籤偵測操作來顯示 Amazon Rekognition Video 在儲存於 Amazon S3 儲存貯體中的影片內偵測標籤 (物件、活動、概念與活動) 的方法。相同的方法適用於其他 Amazon Rekognition Video 作業 — 例如，和. StartFaceDetectionStartPersonTracking 範例 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 說明如何使用 Amazon SQS 佇列自 Amazon SNS 主題取得完成狀態分析以分析影片。同時也用做為其它 Amazon Rekognition Video 範例的基礎，例如 人物路徑。如需 AWS CLI 範例，請參閱使用分析視訊 AWS Command Line Interface。

主題

- 開始影片分析
- 取得 Amazon Rekognition Video 分析請求的完成狀態
- 取得 Amazon Rekognition Video 分析結果

開始影片分析

您可以透過電話啟動 Amazon Rekognition Video 標籤偵測要求。StartLabelDetection以下是由 StartLabelDetection 傳遞的 JSON 請求範例。

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

輸入參數 Video 提供影片檔案名稱以及擷取檔案的 Amazon S3 儲存貯體。NotificationChannel 包含在影片分析請求完成時由 Amazon Rekognition Video 所通知的 Amazon SNS 主題 Amazon Resource Name (ARN)。Amazon SNS 主題必須與您呼叫的 Amazon Rekognition Video 端點位於同

一個 AWS 區域。NotificationChannel 也包含允許 Amazon Rekognition Video 發佈到 Amazon SNS 主題的角色 ARN。透過 IAM 服務角色的建立，您給予 Amazon Rekognition 發佈至 Amazon SNS 主題的許可。如需詳細資訊，請參閱 設定 Amazon Rekognition Video。

您也可以指定選用的輸入參數，JobTag，可讓您找出顯示完成狀態且已發佈至 Amazon SNS 主題的任務。

為避免分析任務發生意外的重複，您可以選擇性地提供等冪符記 ClientRequestToken。如果您提供用於 ClientRequestToken 的值，Start 操作將傳回相同的 JobId 以用於對開始操作執行多個相同的呼叫，例如 StartLabelDetection。ClientRequestToken 符記有 7 天的存留期。在 7 天後，您可以再次使用它。如果您在符記的存留期內重新使用符記，會發生下列情況：

- 如果您使用相同的 Start 操作與相同的輸入參數來重新使用字符，將傳回相同的 JobId。不會再次執行任務，而 Amazon Rekognition Video 也不會傳送完成狀態到已註冊的 Amazon SNS 主題。
- 如果您以相同的 Start 操作搭配些微變更的參數來重新使用符記，您會得到一個 IdempotentParameterMismatchException (HTTP 狀態碼：400) 例外狀況。
- 由於您會從 Amazon Rekognition 取得無法預測的結果，因此您不應使用具有不同 Start 操作的字符。

對於 StartLabelDetection 操作的回應為任務識別碼 (JobId)。使用 JobId 來追蹤請求並在 Amazon Rekognition Video 發佈完成狀態到 Amazon SNS 主題後取得分析結果。例如：

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

如果您同時開始太多任務，呼叫 StartLabelDetection 將引發 LimitExceededException (HTTP 狀態碼：400)，直到數量同時執行任務的數量低於 Amazon Rekognition 服務限制。

如果您發現 LimitExceededException 例外狀況發生且產生活動激增，請考慮使用 Amazon SQS 佇列來管理傳入的請求。如果您發現 Amazon SQS 佇列無法管理平均並行請求數量，且仍然收到LimitExceededException例外狀況，請聯絡 AWS 支援部門。

取得 Amazon Rekognition Video 分析請求的完成狀態

Amazon Rekognition Video 將傳送分析完成通知到已註冊的 Amazon SNS 主題。通知包含任務識別碼和並以 JSON 字串顯示操作的完成狀態。成功的影片分析請求將包含 SUCCEEDED 狀態。例如，以下結果顯示偵測標籤任務已成功處理。

```
{
```

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

如需詳細資訊，請參閱 參考：影片分析結果通知。

若要取得 Amazon Rekognition Video 發佈至 Amazon SNS 主題的狀態資訊，請使用以下其中一個選項：

- AWS Lambda：您可以註冊 AWS Lambda 函數，以寫入 Amazon SNS 主題。當 Amazon Rekognition 通知 Amazon SNS 主題請求已完成時，將呼叫此功能。如果您需要伺服器端程式碼來處理影片分析請求的結果，請使用 Lambda 函數。例如，您可能想要使用伺服器端程式碼來標註影片，或在傳回資訊到客戶端應用程式前針對影片內容建立報告。我們也建議使用伺服器端處理大型影片，因為 Amazon Rekognition API 可能會傳回大量資料。
- Amazon Simple Queue Service：您可以訂閱 Amazon SQS 佇列到 Amazon SNS 主題。接著請輪詢 Amazon SQS 佇列以擷取由 Amazon Rekognition 在影片分析請求完成時發佈的完成狀態。如需詳細資訊，請參閱 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。如果您只想從客戶端應用程式呼叫 Amazon Rekognition Video 操作，請使用 Amazon SQS 佇列。

> **Important**
>
> 我們不建議您透過重複呼叫 Amazon Rekognition Video `Get` 操作來取得請求完成狀態。這是因為如果執行太多請求，Amazon Rekognition Video 將對 `Get` 操作進行節制。如果您同時處理多個影片，監控一個 SQS 佇列的完成通知將會比輪詢 Amazon Rekognition Video 以獲得個別影片的狀態更容易且有效率。

取得 Amazon Rekognition Video 分析結果

若要取得影片分析請求結果，首先請確認自 Amazon SNS 主題擷取的完成狀態為 `SUCCEEDED`。然後呼叫 `GetLabelDetection`，將傳遞自 `StartLabelDetection` 傳回的 `JobId` 值。請求 JSON 格式類似於以下範例：

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

JobId 是視訊分析作業的識別碼。由於影片分析可以產生大量資料，請使用 MaxResults 來指定最大數量的結果，以在單次取得操作中傳回結果。MaxResults 的預設值為 1000。如果您指定的值大於 1000，最多只能傳回 1000 個結果。如果操作不會傳回整組結果，將在操作回應中傳回下一頁的分頁符記。如果您自前一個取得請求中獲得一個分頁符記，請搭配 NextToken 一起使用以取得下一頁結果。

> **Note**
>
> Amazon Rekognition 會保留影片分析操作的結果 7 天。在此時間後您將無法擷取分析結果。

GetLabelDetection 操作回應 JSON 格式類似於以下範例：

```
{
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 60.51791763305664,
                "Parents": [],
                "Name": "Electronics"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Human"
            }
        },
        {
            "Timestamp": 0,
```

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

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

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

GetLabelDetection 和 GetContentModeration 操作可讓您依時間戳記或標籤名稱來排序分析結果。您也可以依影片區段或時間戳記彙總結果。

您可以透過偵測時間 (依據影片開始的毫秒計算) 來排序結果，或依照偵測的實體 (物件、臉部、名人、內容管制標籤或人物) 字母排序。若要依時間排序，將 SortBy 輸入參數值設為 TIMESTAMP。如果未指定 SortBy，預設行為是依據時間排序。前述範例是依時間排序。若要依據實體排序，使用 SortBy 輸入參數搭配適合您要執行的操作值。例如，在對 GetLabelDetection 的呼叫中依偵測的標籤排序時，需使用值 NAME。

若要依時間戳記彙總結果，請將 AggregateBy 參數值設定為 TIMESTAMPS。若要依視訊區段彙總，請將 AggregateBy 的值設定為 SEGMENTS。SEGMENTS 彙總模式將隨著時間的推移彙總標籤，同時使用 2 FPS 採樣和每幀輸出為標籤 TIMESTAMPS 提供偵測到的時間戳 (注意：此當前採樣率可能會發生變化，不應對當前採樣率進行假設)。如未指定任何值，預設為 TIMESTAMPS。

設定 Amazon Rekognition Video

若要在儲存的影片中使用 Amazon Rekognition Video API，您必須設定使用者以及 IAM 服務角色來存取您的 Amazon SNS 主題。您也必須讓 Amazon SQS 佇列訂閱您的 Amazon SNS 主題。

> **Note**
>
> 如果您正依照這些說明設定 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 範例，則無需進行步驟 3、4、5 和 6。此範例包括用於建立並設定 Amazon SNS 主題和 Amazon SQS 佇列的程式碼。

本節中的範例將依據說明來提供 Amazon Rekognition Video 存取多個主題的權限，並以此來建立新的 Amazon SNS 主題。如果您想要使用現有 Amazon SNS 主題，請在步驟 3 使用 提供存取目前 Amazon SNS 主題的權限。

若要設定 Amazon Rekognition Video

1. 設置一個 AWS 帳戶以訪問 Amazon Rekognition Video。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。
2. 安裝並設定必要的 AWS SDK。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。
3. 若要執行此開發人員指南中的程式碼範例，請確定您選擇的使用者具有程式設計存取權。如需詳細資訊，請參閱 授與程式設計存取權。

   您的使用者還需要至少下列許可：

   - 亞馬遜人 FullAccess
   - AmazonRekognitionFullAccess
   - 亞馬遜 FullAccess
   - 亞馬遜 SNS FullAccess

   如果您使用 IAM Identity Center 進行驗證，請將許可新增至角色的權限集，否則將許可新增至 IAM 角色。
4. 使用 Amazon SNS 主控台建立 Amazon SNS 主題。在主題名稱前面加上。AmazonRekognition請記下主題的 Amazon Resource Name (ARN)。請確認此主題與您使用的 AWS 端點位於同一個區域。
5. 使用 Amazon SQS 主控台來建立 Amazon SQS 佇列。記下佇列 ARN。
6. 將佇列訂閱至您在步驟 3 中建立的主題。
7. 將許可提供給 Amazon SNS 主題，以將訊息傳送至 Amazon SQS 佇列。
8. 建立 IAM 服務角色來提供存取您 Amazon SNS 主題的 Amazon Rekognition Video 權限。記下服務角色的 Amazon Resource Name (ARN)。如需詳細資訊，請參閱 提供對多個 Amazon SNS 主題的存取權限。
9. 為了確保您的帳戶安全，您需要將 Rekognition 的存取範圍限制在您正在使用的資源。這可以通過將信任政策附加到您的 IAM 服務角色來完成。如需如何執行此作業的資訊，請參閱 預防跨服務混淆代理人。
10. 將以下內嵌政策新增到您在步驟 1 所建立的使用者：

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

將內嵌政策命名為您想要的名稱。

11. 如果您使用客戶受管 AWS Key Management Service 金鑰來加密 Amazon S3 儲存貯體中的影片，請將許可新增至允許您在步驟 7 中建立的服務角色解密影片的金鑰。服務角色至少需要 kms:GenerateDataKey 和 kms:Decrypt 動作的權限。例如：

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

如需詳細資訊，請參閱 Amazon S3 儲存貯體使用自訂 AWS KMS 金鑰的預設加密。如何允許使用者從儲存貯體下載並上傳至值區？以及搭配存放在 AWS Key Management Service (SSE-KMS) 中的 KMS 金鑰使用伺服器端加密來保護資料。

12. 您現在可執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 和 使用分析視訊 AWS Command Line Interface 中的範例。

提供對多個 Amazon SNS 主題的存取權限

使用 IAM 服務角色來提供存取您所建立的 Amazon SNS 主題的 Amazon Rekognition Video 權限。IAM 提供用於建立 Amazon Rekognition Video 服務角色的 Rekognition 使用案例。

您可以使用AmazonRekognitionServiceRole許可政策並在主題名稱前面加上 — (例如，)，授予 Amazon Amazon Rekognition Video 存取多個 Amazon SNS 主題。AmazonRekognitionAmazonRekognitionMyTopicName

授予 Amazon Rekognition Video 存取多個 Amazon SNS 主題

1. 建立 IAM 服務角色。請使用下列資訊來建立 IAM 服務角色：
    1. 針對服務名稱選擇 Rekognition。
    2. 為服務角色使用案例選擇 Rekognition。您應該會看到列出的AmazonRekognitionServiceRole權限原則。AmazonRekognitionServiceRole提供 Amazon Rekognition Video 存取前綴的 Amazon SNS 主題。AmazonRekognition
    3. 將服務角色命名為您想要的名稱。
2. 請記下服務角色的 ARN。您需要此資訊才可開始影片分析操作。

提供存取目前 Amazon SNS 主題的權限

您可以建立許可政策，以允許 Amazon Rekognition Video 存取現有的 Amazon SNS 主題。

提供現有 Amazon SNS 主題的 Amazon Rekognition Video 存取權

1. 使用 IAM JSON 政策編輯器建立新的許可政策，並使用下列政策。以想要的 Amazon SNS 主題 Amazon Resource Name (ARN) 來取代 topicarn。

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

```
}
```

2. 建立 IAM 服務角色，或更新現有的 IAM 服務角色。請使用下列資訊來建立 IAM 服務角色：
   1. 針對服務名稱選擇 Rekognition。
   2. 為服務角色使用案例選擇 Rekognition。
   3. 連接您在步驟 1 中建立的許可政策。
3. 請記下服務角色的 ARN。您需要此資訊才可開始影片分析操作。

使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片

此程序說明如何使用 Amazon Rekognition Video 標籤偵測操作、儲存在 Amazon S3 儲存貯體中的影片、以及 Amazon SNS 主題等方法來偵測影片中的標籤。程序同時也將說明如何使用 Amazon SQS 佇列從 Amazon SNS 主題取得完成狀態。如需詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。不限於使用 Amazon SQS 佇列。例如，您可以使用 AWS Lambda 函數來取得完成狀態。如需詳細資訊，請參閱使用 Amazon SNS 通知來呼叫 Lambda 函數。

此程序中的範例程式碼說明如何執行下列動作：

1. 建立 Amazon SNS 主題。
2. 建立 Amazon SQS 佇列。
3. 給予 Amazon Rekognition Video 發佈影片分析操作完成狀態到 Amazon SNS 主題的許可。
4. 訂閱佇列至 Amazon SNS 主題。
5. 通過呼叫啟動視頻分析請求StartLabelDetection。
6. 從 Amazon SQS 佇列取得完成狀態。此範例追蹤在 `StartLabelDetection` 傳回的任務識別碼 (`JobId`)，在自完成狀態中讀取後，只會顯示相符的工作識別碼之結果。如果其他應用程式使用相同的佇列和主題，這便是項重要的考量條件。為了方便起見，此範例刪除不符合的任務。考慮將它們新增到 Amazon SQS 無效字母佇列以供進一步調查。
7. 通過呼叫獲取並顯示視頻分析結果GetLabelDetection。

必要條件

此程序的範例程式碼以 Java 和 Python 提供。您需要安裝適當的 AWS SDK。如需詳細資訊，請參閱 Amazon Rekognition 入門。您使用的 AWS 帳戶必須有 Amazon Rekognition API 的存取許可。如需詳細資訊，請參閱 Amazon Rekognition 定義的動作。

若要偵測影片中的標籤

1. 設定使用者對 Amazon Rekognition Video 的存取權限，並設定 Amazon Rekognition Video 對 Amazon SNS 的存取。如需詳細資訊，請參閱 設定 Amazon Rekognition Video。由於範例程式碼會建立並設定 Amazon SNS 主題和 Amazon SQS 佇列，因此無須進行步驟 3、4、5 和 6。
2. 將 MOV 或 MPEG-4 格式的影片上傳到 Amazon S3 儲存貯體。為達測試目的，請上傳長度不超過 30 秒的影片。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用以下程式碼範例來偵測影片中的標籤。

   Java

   在函數 `main` 中：

   - 將 `roleArn` 取代為您在 若要設定 Amazon Rekognition Video 步驟 7 建立的 IAM 服務角色 ARN。
   - 以您在步驟 2 中指定的儲存貯體與影片檔名稱來取代 `bucket` 與 `video` 的值。

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

package com.amazonaws.samples;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Condition;
import com.amazonaws.auth.policy.Principal;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.SQSActions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CelebrityDetail;
import com.amazonaws.services.rekognition.model.CelebrityRecognition;
import com.amazonaws.services.rekognition.model.CelebrityRecognitionSortBy;
import com.amazonaws.services.rekognition.model.ContentModerationDetection;
import com.amazonaws.services.rekognition.model.ContentModerationSortBy;
```

```
import com.amazonaws.services.rekognition.model.Face;
import com.amazonaws.services.rekognition.model.FaceDetection;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.FaceSearchSortBy;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.GetContentModerationRequest;
import com.amazonaws.services.rekognition.model.GetContentModerationResult;
import com.amazonaws.services.rekognition.model.GetFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.GetFaceDetectionResult;
import com.amazonaws.services.rekognition.model.GetFaceSearchRequest;
import com.amazonaws.services.rekognition.model.GetFaceSearchResult;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
import com.amazonaws.services.rekognition.model.GetPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.GetPersonTrackingResult;
import com.amazonaws.services.rekognition.model.Instance;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.services.rekognition.model.LabelDetection;
import com.amazonaws.services.rekognition.model.LabelDetectionSortBy;
import com.amazonaws.services.rekognition.model.NotificationChannel;
import com.amazonaws.services.rekognition.model.Parent;
import com.amazonaws.services.rekognition.model.PersonDetection;
import com.amazonaws.services.rekognition.model.PersonMatch;
import com.amazonaws.services.rekognition.model.PersonTrackingSortBy;
import com.amazonaws.services.rekognition.model.S3Object;
import
 com.amazonaws.services.rekognition.model.StartCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.StartCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.StartContentModerationRequest;
import com.amazonaws.services.rekognition.model.StartContentModerationResult;
import com.amazonaws.services.rekognition.model.StartFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.StartFaceDetectionResult;
import com.amazonaws.services.rekognition.model.StartFaceSearchRequest;
import com.amazonaws.services.rekognition.model.StartFaceSearchResult;
import com.amazonaws.services.rekognition.model.StartLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.StartLabelDetectionResult;
import com.amazonaws.services.rekognition.model.StartPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.StartPersonTrackingResult;
import com.amazonaws.services.rekognition.model.Video;
import com.amazonaws.services.rekognition.model.VideoMetadata;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
```

```
import com.amazonaws.services.sns.model.CreateTopicResult;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.*;

public class VideoDetect {


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

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


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

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

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

        CreateTopicandQueue();
```

```
        //=================================================

        StartLabelDetection(bucket, video);

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

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


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

    }


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


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

        //loop until the job status is published. Ignore other messages in
queue.
        do{
            messages = sqs.receiveMessage(sqsQueueUrl).getMessages();
            if (dotLine++<40){
                System.out.print(".");
            }else{
                System.out.println();
                dotLine=0;
            }

            if (!messages.isEmpty()) {
                //Loop through messages received.
                for (Message message: messages) {
                    String notification = message.getBody();

                    // Get status and job id from notification.
```

```
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found was " + operationJobId);
                    // Found job. Get the results and display.
                    if(operationJobId.asText().equals(startJobId)){
                        jobFound=true;
                        System.out.println("Job id: " + operationJobId );
                        System.out.println("Status : " +
operationStatus.toString());
                        if (operationStatus.asText().equals("SUCCEEDED")){
                         success=true;
                        }
                        else{
                            System.out.println("Video analysis failed");
                        }


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

                    else{
                        System.out.println("Job received was not job " +
startJobId);
                        //Delete unknown message. Consider moving message to
dead letter queue

sqs.deleteMessage(sqsQueueUrl,message.getReceiptHandle());
                    }
                }
            }
            else {
                Thread.sleep(5000);
            }
        } while (!jobFound);

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

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

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


       StartLabelDetectionRequest req = new StartLabelDetectionRequest()
               .withVideo(new Video()
                       .withS3Object(new S3Object()
                               .withBucket(bucket)
                               .withName(video)))
               .withMinConfidence(50F)
               .withJobTag("DetectingLabels")
               .withNotificationChannel(channel);

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

   }

   private static void GetLabelDetectionResults() throws Exception{

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

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

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

```
            labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

            VideoMetadata videoMetaData=labelDetectionResult.getVideoMetadata();

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


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

            for (LabelDetection detectedLabel: detectedLabels) {
                long seconds=detectedLabel.getTimestamp();
                Label label=detectedLabel.getLabel();
                System.out.println("Millisecond: " + Long.toString(seconds) + "
");

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

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

```
                        System.out.println("        " + parent.getName());
                    }
                }
                System.out.println();
            }
        } while (labelDetectionResult !=null &&
labelDetectionResult.getNextToken() != null);

   }

   // Creates an SNS topic and SQS queue. The queue is subscribed to the
topic.
   static void CreateTopicandQueue()
   {
       //create a new SNS topic
       snsTopicName="AmazonRekognitionTopic" +
Long.toString(System.currentTimeMillis());
       CreateTopicRequest createTopicRequest = new
CreateTopicRequest(snsTopicName);
       CreateTopicResult createTopicResult =
sns.createTopic(createTopicRequest);
       snsTopicArn=createTopicResult.getTopicArn();

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

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

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

```
                    );


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


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

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

Python

在函數 `main` 中：

- 將 `roleArn` 取代為您在 若要設定 Amazon Rekognition Video 步驟 7 建立的 IAM 服務角色 ARN。
- 以您在步驟 2 中指定的儲存貯體與影片檔名稱來取代 `bucket` 與 `video` 的值。
- 將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。
- 您還可以在設定參數中包括過濾條件。例如，您可以在所需值的清單之外，邊使用 `LabelsInclusionFilter` 或 `LabelsExclusionFilter`。在下面的程式碼中，您可以取消註釋 `Features` 和 `Settings` 部分，並提供自己的值，以將傳回的結果限制為只有您感興趣的標籤。

- 在呼叫 GetLabelDetection 中，您可以提供 SortBy 和 AggregateBy 引數的值。若要依時間排序，將 SortBy 輸入參數值設為 TIMESTAMP。若要依據實體排序，使用 SortBy 輸入參數搭配適合您要執行的操作值。若要依時間戳記彙總結果，請將 AggregateBy 參數值設定為 TIMESTAMPS。若要依影片片段彙總，請使用 SEGMENTS。

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

import boto3
import json
import sys
import time

class VideoDetect:

    jobId = ''

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

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

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

    def GetSQSMessageSuccess(self):

        jobFound = False
        succeeded = False

        dotLine = 0
```

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

            if sqsResponse:

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

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

                        self.sqs.delete_message(QueueUrl=self.sqsQueueUrl,

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

ReceiptHandle=message['ReceiptHandle'])

        return succeeded
```

```
    def StartLabelDetection(self):
        response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},

 NotificationChannel={'RoleArn': self.roleArn,

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

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

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

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

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

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

                print("Timestamp: " + str(labelDetection['Timestamp']))
```

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

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

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

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

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

    def CreateTopicandQueue(self):

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

        # Create SNS topic

        snsTopicName = "AmazonRekognitionExample" + millis

        topicResponse = self.sns.create_topic(Name=snsTopicName)
```

```
        self.snsTopicArn = topicResponse['TopicArn']

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

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

        sqsQueueArn = attribs['QueueArn']

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

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

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

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

def main():

    roleArn = 'role-arn'
    bucket = 'bucket-name'
    video = 'video-name'

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

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

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

    analyzer.DeleteTopicandQueue()

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

Node.Js

請參閱以下範本程式碼：

- 將 REGION 的值取代為您帳戶營運地區的名稱。
- 以包含影片檔案的 Amazon S3 儲存貯體之名稱取代 bucket 的值。
- 在 Amazon S3 儲存貯體中，以您的 S3 儲存貯體名稱取代 videoName 的值。
- 將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。
- 將 roleArn 取代為您在 若要設定 Amazon Rekognition Video 步驟 7 建立的 IAM 服務角色 ARN。

```
import { CreateQueueCommand, GetQueueAttributesCommand, GetQueueUrlCommand,
  SetQueueAttributesCommand, DeleteQueueCommand, ReceiveMessageCommand,
 DeleteMessageCommand } from  "@aws-sdk/client-sqs";
import {CreateTopicCommand, SubscribeCommand, DeleteTopicCommand } from "@aws-
sdk/client-sns";
import  { SQSClient } from "@aws-sdk/client-sqs";
import  { SNSClient } from "@aws-sdk/client-sns";
import  { RekognitionClient, StartLabelDetectionCommand,
 GetLabelDetectionCommand } from "@aws-sdk/client-rekognition";
import { stdout } from "process";
import {fromIni} from '@aws-sdk/credential-providers';

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

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

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

// Set the parameters
const sqsParams = {
  QueueName: sqsQueueName, //SQS_QUEUE_URL
  Attributes: {
    DelaySeconds: "60", // Number of seconds delay.
    MessageRetentionPeriod: "86400", // Number of seconds delay.
  },
```

```
};

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

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

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

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

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

  // Begin retrieving label detection results
  while (finished == false){
    var response = await rekClient.send(new GetLabelDetectionCommand({JobId:
 startJobId, MaxResults: maxResults,
     NextToken: paginationToken, SortBy:'TIMESTAMP'}))
     // Log metadata
     console.log(`Codec: ${response.VideoMetadata.Codec}`)
     console.log(`Duration: ${response.VideoMetadata.DurationMillis}`)
     console.log(`Format: ${response.VideoMetadata.Format}`)
     console.log(`Frame Rate: ${response.VideoMetadata.FrameRate}`)
```

```
      console.log()
      // For every detected label, log label, confidence, bounding box, and
 timestamp
      response.Labels.forEach(labelDetection => {
        var label = labelDetection.Label
        console.log(`Timestamp: ${labelDetection.Timestamp}`)
        console.log(`Label: ${label.Name}`)
        console.log(`Confidence: ${label.Confidence}`)
        console.log("Instances:")
        label.Instances.forEach(instance =>{
          console.log(`Confidence: ${instance.Confidence}`)
          console.log("Bounding Box:")
          console.log(`Top: ${instance.Confidence}`)
          console.log(`Left: ${instance.Confidence}`)
          console.log(`Width: ${instance.Confidence}`)
          console.log(`Height: ${instance.Confidence}`)
          console.log()
        })
      console.log()
      // Log parent if found
      console.log("   Parents:")
      label.Parents.forEach(parent =>{
        console.log(`    ${parent.Name}`)
      })
      console.log()
      // Searh for pagination token, if found, set variable to next token
      if (String(response).includes("NextToken")){
        paginationToken = response.NextToken

      }else{
        finished = true
      }

      })
  }
}

// Checks for status of job completion
const getSQSMessageSuccess = async(sqsQueueUrl, startJobId) => {
  try {
    // Set job found and success status to false initially
    var jobFound = false
    var succeeded = false
    var dotLine = 0
```

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

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

```
          var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
        }
      }
    }
  return succeeded
  } catch(err) {
    console.log("Error", err);
  }
};

// Start label detection job, sent status notification, check for success
 status
// Retrieve results if status is "SUCEEDED", delete notification queue and
 topic
const runLabelDetectionAndGetResults = async () => {
  try {
    const sqsAndTopic = await createTopicandQueue();
    const startLabelDetectionRes = await startLabelDetection(roleArn,
 sqsAndTopic[1]);
    const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
 startLabelDetectionRes)
    console.log(getSQSMessageSuccess)
    if (getSQSMessageSuccess){
      console.log("Retrieving results:")
      const results = await getLabelDetectionResults(startLabelDetectionRes)
    }
    const deleteQueue = await sqsClient.send(new DeleteQueueCommand({QueueUrl:
 sqsAndTopic[0]}));
    const deleteTopic = await snsClient.send(new DeleteTopicCommand({TopicArn:
 sqsAndTopic[1]}));
    console.log("Successfully deleted.")
  } catch (err) {
    console.log("Error", err);
  }
};

runLabelDetectionAndGetResults()
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

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

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

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

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

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

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

    SqsClient sqs = SqsClient.builder()
        .region(Region.US_WEST_2)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

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

    startLabels(rekClient, channel, bucket, video);
    getLabelJob(rekClient, sqs,  queueUrl);
```

```
    System.out.println("This example is done!");
    sqs.close();
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_detect.main]
public static void startLabels(RekognitionClient rekClient,
                               NotificationChannel channel,
                               String bucket,
                               String video) {
    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();

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

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

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

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

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

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

            if (status.compareTo("SUCCEEDED") == 0)
                ans = false;
            else
                System.out.println(yy +" status is: "+status);

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

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

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

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

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

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

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

                // Get the status and job id from the notification
                ObjectMapper mapper = new ObjectMapper();
                JsonNode jsonMessageTree = mapper.readTree(notification);
                JsonNode messageBodyText = jsonMessageTree.get("Message");
                ObjectMapper operationResultMapper = new ObjectMapper();
                JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                JsonNode operationJobId = jsonResultTree.get("JobId");
```

```
                JsonNode operationStatus = jsonResultTree.get("Status");
                System.out.println("Job found in JSON is " + operationJobId);

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

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

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

                    sqs.deleteMessage(deleteMessageRequest);
                }

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

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

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

    int maxResults=10;
```

```
    String paginationToken=null;
    GetLabelDetectionResponse labelDetectionResult=null;

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


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

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

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

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

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

                if (instances.isEmpty()) {
                    System.out.println("        " + "None");
                } else {
                    for (Instance instance : instances) {
                        System.out.println("        Confidence: " +
instance.confidence().toString());
```

```
                        System.out.println("        Bounding box: " +
instance.boundingBox().toString());
                    }
                }
                System.out.println("   Parent labels for " + label.name() +
":");
                List<Parent> parents = label.parents();

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

    } catch(RekognitionException e) {
        e.getMessage();
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.recognize_video_detect.main]
}
```

4. 建置並執行程式碼。此操作可能需要一些時間來完成。完成之後，將顯示在影片中偵測到的標籤清單。如需詳細資訊，請參閱 偵測影片中的標籤。

使用分析視訊 AWS Command Line Interface

您可以使用 AWS Command Line Interface (AWS CLI) 呼叫 Amazon Rekognition Video 操作。設計模式與將 Amazon Rekognition Video API 與 AWS SDK for Java 或其他 AWS 開發套件搭配使用相同。如需詳細資訊，請參閱 Amazon Rekognition Video API 概觀。下列程序說明如何使 AWS CLI 用偵測視訊中的標籤。

呼叫 `start-label-detection` 來開始偵側影片中的標籤。當 Amazon Rekognition 完成影片分析時，完成狀態會傳送到 `start-label-detection` 的 `--notification-channel` 參數中所指定的 Amazon SNS 主題。您可以透過訂閱 Amazon Simple Queue Service (Amazon SQS) 佇列到 Amazon SNS 主題來取得完成狀態。接著輪詢接收訊息，以自 Amazon SQS 佇列取得完成狀態。

呼叫 StartLabelDetection 時，您可以透過向 LabelsInclusionFilter 和/或 LabelsExclusionFilter 參數提供過濾參數來過濾結果。如需詳細資訊，請參閱偵測影片中的標籤。

完成狀態通知的為 receive-message 回應內的 JSON 結構。您需要自回應中擷取出 JSON。如需完成狀態 JSON 的相關資訊，請參閱 參考：影片分析結果通知。如果已完整狀態 JSON 的 Status 欄位值為 SUCCEEDED，您可以呼叫 get-label-detection 來取得影片分析請求的結果。呼叫 GetLabelDetection 時，您可以使用 SortBy 和 AggregateBy 引數來排序和彙總傳回的結果。

以下程序不包含用以輪詢 Amazon SQS 佇列的程式碼。此外也不包含用於剖析自 Amazon SQS 佇列傳回的 JSON 之程式碼。如需 Java 範例，請參閱 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。

必要條件

要運行此過程，你需要安 AWS CLI 裝。如需詳細資訊，請參閱 Amazon Rekognition 入門。您使用的 AWS 帳戶必須有 Amazon Rekognition API 的存取許可。如需詳細資訊，請參閱 Amazon Rekognition 定義的動作。

若要設定 Amazon Rekognition Video 並上傳影片

1. 設定使用者對 Amazon Rekognition Video 的存取權限，並設定 Amazon Rekognition Video 對 Amazon SNS 的存取。如需詳細資訊，請參閱 設定 Amazon Rekognition Video。
2. 將 MOV 或 MPEG-4 格式的影片上傳到 S3 儲存貯體。在開發和測試階段，我們建議使用長度不超過 30 秒的短片。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

若要偵測影片中的標籤

1. 執行下列 AWS CLI 命令以開始偵測視訊中的標籤。

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

更新下列的值：

- 將 bucketname 與 videofile 變更為您在步驟 2 中指定的 Amazon S3 儲存貯體與文檔名稱。
- 將 us-east-1 變更為您正在使用的 AWS 區域。
- 將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。
- 將 TopicARN 變更為您在 設定 Amazon Rekognition Video 步驟 3 建立的 Amazon SNS 主題 ARN。
- 將 RoleARN 變更為您在步驟 7 建立的 設定 Amazon Rekognition Video IAM 服務角色的 ARN。
- 如有需要，可以指定 endpoint-url。AWS CLI 應根據提供的區域自動判斷適當的端點 URL。但是，如果您使用私有 VPC 中的端點，則可能需要指定 endpoint-url。AWS 服務端點資源會列出指定端點 URL 的語法，以及每個區域的名稱和程式碼。
- 您還可以在設定參數中包括過濾條件。例如，您可以在所需值的清單之外，邊使用 LabelsInclusionFilter 或 LabelsExclusionFilter。

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。如需範例，請參閱下列內容：

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

2. 請記下回應中的 JobId 值。回應看起來類似以下 JSON 範例。

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

3. 編寫程式碼來輪詢 Amazon SQS 佇列以取得完成狀態 JSON (使用接收訊息)。
4. 編寫程式碼以自完成狀態 JSON 擷取 Status 欄位。
5. 如果的值Status為SUCCEEDED，請執行下列 AWS CLI 命令以顯示標籤偵測結果。

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

更新下列的值：

- 變更 JobId 以符合您在步驟 2 中所記下的任務識別碼。
- 變更 Endpoint 和 us-east-1 為 AWS 端點與您正在使用的區域。

結果看起來類似以下 JSON 範例：

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

參考：影片分析結果通知

Amazon Rekognition 將 Amazon Rekognition Video 分析請求的結果 (包括完成狀態) 發佈至 Amazon Simple Notification Service (Amazon SNS) 主題。若要從 Amazon SNS 主題取得通知，請使用 Amazon 簡單佇列服務佇列或 AWS Lambda 函數。如需詳細資訊，請參閱 the section called “呼叫 Amazon Rekognition Video 操作”。如需範例，請參閱使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。

承載內容以下方 JSON 格式顯示：

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

名稱	描述
JobId	任務識別碼。符合從作業傳回的工Start作識別碼，例如StartPersonTracking。
Status	任務的狀態。有效值為「SUCCEEDED」(成功)、「FAILED」(失敗) 或「ERROR」(錯誤)。
API	用於分析輸入影片的 Amazon Rekognition Video 操作。
JobTag	任務的識別碼。您可以JobTag在呼叫「開始」作業中指定，例如StartLabelDetection。
時間戳記	顯示任務完成的 Unix 時間戳記。
影片	已完成處理的影片詳細資訊。包含檔案名稱與檔案所存放的 Amazon S3 儲存貯體。

以下為傳送到 Amazon SNS 主題的成功通知範例。

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

對 Amazon Rekognition Video 進行疑難排解

以下內容涵蓋使用 Amazon Rekognition Video 與已儲存影片時的疑難排解資訊。

我都沒有收到傳送至 Amazon SNS 主題的完成狀態訊息

當影片分析完成時，Amazon Rekognition Video 會將狀態資訊發佈至 Amazon SNS 主題。一般而言，您可以使用 Amazon SQS 佇列或 Lambda 函數來訂閱主題，以取得完成狀態訊息。為順利進行調查，您可以透過電子郵件訂閱 Amazon SNS 主題。如此，電子郵件收件匣即會收到傳送至 Amazon SNS 主題的訊息。如需詳細資訊，請參閱訂閱 Amazon SNS 主題。

如果您的應用程式沒有收到該訊息，請考慮下列情況：

- 請確認分析已完成。請檢查取得操作中的 `JobStatus` 值 (例如，`GetLabelDetection`)。如果該值為 `IN_PROGRESS`，表示分析尚未完成，而完成狀態也尚未發布至 Amazon SNS 主題。
- 請確認您擁有 IAM 服務角色，以授予 Amazon Rekognition Video 發佈至 Amazon SNS 主題的許可。如需詳細資訊，請參閱 設定 Amazon Rekognition Video。
- 確認您正在使用的 IAM 服務角色可以使用角色憑證發佈到 Amazon SNS 主題，並確認服務角色的許可範圍安全地限於您正在使用的資源。執行以下步驟：
  - 取得使用者的 Amazon Resource Name (ARN)：

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

  - 將使用者 ARN 新增至角色信任關係。如需詳細資訊，請參閱修改角色。下列範例信任原則會指定使用者的角色憑證，並將服務角色的權限限制為您正在使用的資源 (如需有關安全地限制服務角色權限範圍的詳細資訊，請參閱 預防跨服務混淆代理人)：

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

- 假設該角色：`aws sts assume-role --role-arn arn:Role ARN --role-session-name SessionName --profile RekognitionUser`
- 發佈至 Amazon SNS 主題。`aws sns publish --topic-arn arn:Topic ARN --message "Hello World!" --region us-east-1 --profile RekognitionUser`

如果 AWS CLI 命令有效，則您會收到該消息（如果您已通過電子郵件訂閱該主題，則在電子郵件收件箱中）。如果您沒有收到該訊息：

- 確認您已設定 Amazon Rekognition Video。如需詳細資訊，請參閱 設定 Amazon Rekognition Video。
- 請查看對此問題執行故障排除的其他秘訣。

- 確認您使用的是正確的 Amazon SNS 主題：
  - 如果您使用 IAM 服務角色來授予 Amazon Rekognition Video 存取單一 Amazon SNS 主題的權限，請確認取得許可的 Amazon SNS 主題正確無誤。如需詳細資訊，請參閱 提供存取目前 Amazon SNS 主題的權限。

- 如果您使用 IAM 服務角色授予 Amazon Rekognition Video 存取多個 SNS 主題的權限，請確認您使用的是正確的主題，且主題名稱前面加上。AmazonRekognition如需詳細資訊，請參閱 提供對多個 Amazon SNS 主題的存取權限。
- 如果您使用 AWS Lambda 函數，請確認您的 Lambda 函數已訂閱正確的 Amazon SNS 主題。如需更多資訊，請參閱 Fanout 至 Lambda 函數 。
- 如果 Amazon SQS 佇列成功訂閱 Amazon SNS 主題，請確定 Amazon SNS 主題擁有傳送訊息至 Amazon SQS 佇列的許可。如需更多資訊，請參閱提供許可給 Amazon SNS 主題，以傳送訊息至 Amazon SQS 佇列。

我需要其他協助針對 Amazon SNS 主題進行疑難排解

您可以 AWS X-Ray 搭配 Amazon SNS 使用，追蹤和分析透過應用程式傳送的訊息。如需詳細資訊，請參閱 Amazon SNS 和 AWS X-Ray.

如需其他協助，您可以將您的問題張貼到 Amazon Rekognition 論壇，或考慮註冊取得AWS 技術支援。

使用串流視訊事件

您可以使用 Amazon Rekognition 影片偵測和辨識人臉，或偵測串流視訊中的物件。亞馬遜重新認知影片使用 Amazon Kinesis 影片串流來接收和處理視訊串流。您可以使用參數建立串流處理器，以顯示您希望串流處理器從視訊串流偵測到的內容。Rekognition 會以 Amazon SNS 和 Amazon S3 通知的形式，從串流影片事件傳送標籤偵測結果。重新認知會將臉部搜尋結果輸出至 Kinesis 資料串流。

臉部搜索流處理器使用`FaceSearchSettings`以搜尋集合中的臉孔。如需如何實作臉部搜尋串流處理器以分析串流視訊中人臉的詳細資訊，請參閱the section called “在串流影片中搜尋集合中的人臉”。

標籤檢測流處理器使用`ConnectedHomeSettings`在流媒體視頻事件中搜索人物，包裹和寵物。如需如何實作標籤偵測串流處理器的詳細資訊，請參閱the section called “偵測串流視訊事件中的標籤”。

亞馬遜視訊串流處理器操作概觀

您可以啟動 Amazon Rekognition 視訊串流處理器，然後將視訊串流至 Amazon Rekognition 視訊，開始分析串流視訊。Amazon Rekognition 視訊串流處理器可讓您啟動、停止和管理串流處理器。您可以透過呼叫 CreateStreamProcessor 來建立串流處理器。用於建立臉部搜尋串流處理器的請求參數包括 Kinesis 影片串流的 Amazon 資源名稱 (ARN)、Kinesis 資料串流，以及用於辨識串流視訊中人臉的集合識別碼。用於建立安全監控串流處理器的請求參數包括 Kinesis 影片串流的 Amazon 資源名稱

(ARN) 和 Amazon SNS 主題、要在影片串流中偵測到的物件類型，以及輸出結果之 Amazon S3 儲存貯體的資訊。您也可以加入您為串流處理器指定的名稱。

您可以透過呼叫 StartStreamProcessor 操作來開始處理影片。若要取得串流處理器的狀態資訊，請呼叫 DescribeStreamProcessor。您可以調用的其他操作是TagResource標記流處理器和DeleteStreamProcessor刪除流處理器。如果您使用的是臉部搜索流處理器，也可以使用StopStreamProcessor停止流處理器。若要取得您帳戶中的串流處理器清單，請呼叫 ListStreamProcessors。

串流處理器開始執行之後，您可以透過您在中指定的 Kinesis 視訊串流將視訊串流至 Amazon Rekognition 影片`CreateStreamProcessor`。您可以使用室壁運動影片串流 SDKPutMedia操作將視頻傳遞到 Kinesis 視頻流。如需範例，請參閱PutMediaAPI 範例。

如需應用程式如何從臉部搜尋串流處理器取用 Amazon Rekognition 視訊分析結果的相關資訊，請參閱讀取串流影片分析結果。

標記亞馬遜視訊串流處理器

您可以使用標籤來識別、組織、搜尋和篩選 Amazon Rekognition 串流處理器。每個標籤都是由使用者定義的津要和值組成的標籤。

主題

- 將標籤新增至新的串流處理器
- 將標籤新增至現有的串流處理器
- 列出串流處理器中的標籤
- 從串流處理器刪除標籤

將標籤新增至新的串流處理器

您可以在建立串流處理器時，將標記新增至串流處理器`CreateStreamProcessor`操作。指定一個或多個標籤`Tags`陣列輸入參數。以下是一個 JSON 範例`CreateStreamProcessor`請求與標籤。

```
{
       "Name": "streamProcessorForCam",
       "Input": {
              "KinesisVideoStream": {
                     "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnnn:stream/
inputVideo"
```

```
                }
        },
        "Output": {
                "KinesisDataStream": {
                        "Arn": "arn:aws:kinesis:us-east-1:nnnnnnnnnnnn:stream/outputData"
                }
        },
        "RoleArn": "arn:aws:iam::nnnnnnnnnnn:role/roleWithKinesisPermission",
        "Settings": {
                "FaceSearch": {
                        "CollectionId": "collection-with-100-faces",
                        "FaceMatchThreshold": 85.5
                },
                "Tags": {
       "Dept": "Engineering",
         "Name": "Ana Silva Carolina",
         "Role": "Developer"
        }
}
```

將標籤新增至現有的串流處理器

若要將一或多個標籤新增至現有的串流處理器，請使用TagResource操作。指定串流處理器的亞馬遜資源名稱 (ARN) (ResourceArn) 和標籤 (Tags) 您要新增的。下面的例子演示了如何添加兩個標籤。

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

> **Note**
>
> 如果您不知道串流處理器的 Amazon 資源名稱，可以使用DescribeStreamProcessor操作。

列出串流處理器中的標籤

若要列出連接到串流處理器的標籤，請使用ListTagsForResource操作並指定流處理器的 ARN (ResourceArn). 響應是連接到指定流處理器的標籤鍵和值的映射。

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

輸出顯示連接到流處理器的標籤列表：

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

從串流處理器刪除標籤

若要從串流處理器移除一或多個標籤，請使用UntagResource操作。指定模型的 ARN (ResourceArn) 和標籤鍵 (Tag-Keys) 您要刪除的。

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

或者，您可以使用以下格式指定標籤鍵：

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

錯誤處理

本節說明執行時間錯誤和其處理方式。同時也說明 Amazon Rekognition 特有的錯誤訊息和代碼。

主題

- 錯誤元件
- 錯誤訊息和錯誤碼
- 應用程式中的錯誤處理

錯誤元件

當您的程式傳送請求時，Amazon Rekognition 會嘗試處理該請求。如果要求成功，亞馬遜會傳回 HTTP 成功狀態碼 (200 OK)，以及要求作業的結果。

如果請求不成功，Amazon Rekognition 會傳回錯誤訊息。每個錯誤都會有三個元件：

- HTTP 狀態碼 (例如 400)。
- 例外狀況名稱 (例如 InvalidS3ObjectException)。
- 錯誤訊息 (例如 Unable to get object metadata from S3. Check object key, region and/or access permissions.)。

AWS 開發套件會負責將錯誤傳播至應用程式，讓您可以採取適當的動作。例如，在 Java 程式中，您可以撰寫 try-catch 邏輯來處理 ResourceNotFoundException。

如果您不使用 AWS 開發套件，則需要剖析來自 Amazon Rekognition 的低階回應內容。以下是這類回應的範例：

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

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

錯誤訊息和錯誤碼

以下是 Amazon Rekognition 傳回的例外狀況清單，依 HTTP 狀態碼分組。確定要重試嗎？為是，則可以重新提交相同的請求。確定要重試嗎？為否，則需要先修正用戶端上的問題，再提交新的請求。

HTTP 狀態碼 400

HTTP 400 狀態碼表示您的請求有問題。問題的一些範例為驗證失敗、遺失的必要參數，或是超過作業的佈建輸送量。您必須先修正應用程式中的問題，再重新提交請求。

AccessDeniedException

訊息:發生錯誤 (AccessDeniedException) 呼叫<Operation>作業時：User: <User ARN>未授權在<Operation>資源上執行:<Resource ARN>。

您未獲授權執行動作。使用已獲授權之使用者或 IAM 角色的 Amazon Resource Name (ARN) 來執行操作。

OK to retry? (確定要重試嗎？) 否

GroupFacesInProgressException

訊息:排程失敗GroupFaces工作。此集合有一個現有的組面工作。

請在現有任務完成後重試操作。

OK to retry? (確定要重試嗎？) 否

IdempotentParameterMismatchException

訊息:該ClientRequestToken：<Token>您提供的已經在使用中。

一個ClientRequestTokeninput 參數已與作業一起重複使用，但至少有一個其他輸入參數與上一次呼叫作業不同。

OK to retry? (確定要重試嗎？) 否

ImageTooLargeException

訊息：影像大小太大。

輸入影像大小超過允許的限制。如果你打電話DetectProtectiveEquipment，影像大小或解析度超過允許的限制。如需詳細資訊，請參閱Amazon Rekognition 中的準則和配額。

OK to retry? (確定要重試嗎？) 否

InvalidImageFormatException

訊息：請求的影像格式無效。

不支援所提供的影像格式。請使用支援的影像格式 (.JPEG 和 .PNG)。如需詳細資訊，請參閱Amazon Rekognition 中的準則和配額。

OK to retry? (確定要重試嗎？) 否

InvalidPaginationTokenException

訊息

- 無效的字符
- 無效的分頁符記

請求中的分頁符記無效。符記可能已過期。

OK to retry? (確定要重試嗎？) 否

InvalidParameterException

訊息：請求有無效的參數。

輸入參數違反限制。請驗證您的參數，然後再次呼叫 API 操作。

OK to retry? (確定要重試嗎？) 否

無效 3ObjectException

訊息：

- 請求有無效的 S3 物件。
- 無法從 S3 取得物件中繼資料。檢查物件金鑰、地區和/或存取權限。

亞馬遜重新認知無法存取請求中指定的 S3 物件。如需詳細資訊，請參閱設定對 S3 的存取：AWS S3 管理存取。如需故障診斷資訊，請參閱故障診斷 Amazon S3。

OK to retry? (確定要重試嗎？) 否

LimitExceededException

訊息：

- 超出帳戶的串流處理器限制，限制 - <目前限制>。

- <開啟任務數> 開啟任務，適用於使用者 <使用者 ARN> 最大限制：<最大限制>

超出亞馬遜重新認知服務限制。例如，如果您同時啟動太多 Amazon Rekognition 視訊任務，請呼叫啟動作業，例如`StartLabelDetection`，提高一個`LimitExceededException`例外狀況 (HTTP 狀態碼：400)，直到同時執行的任務數目低於 Amazon 重新認知服務限制為止。

OK to retry? (確定要重試嗎？) 否

ProvisionedThroughputExceededException

訊息：

- 超過佈建率。
- 超過 S3 下載限制。

請求數超過您的傳輸量限制。如需詳細資訊，請參閱〈亞馬遜重新認知服務限制。

要請求提高限制，請按照以下說明進行操作：the section called “建立案例以變更 TPS 配額”。

OK to retry? (確定要重試嗎？) 是

ResourceAlreadyExistsException

訊息：集合 ID：<集合 ID> 已存在。

已存在使用指定 ID 的集合。

OK to retry? (確定要重試嗎？) 否

ResourceInUseException

訊息：

- 串流處理器名稱已在使用中。
- 指定的資源已在使用中。
- 處理器無法用於停止串流。
- 無法刪除串流處理器。

當資源可用時重試。

OK to retry? (確定要重試嗎？) 否

ResourceNotFoundException

訊息：根據 API 呼叫的不同訊息。

指定的資源不存在。

OK to retry? (確定要重試嗎？) 否

ThrottlingException

訊息：減慢；請求率突然增加。

您的請求率增加過快。請減慢並逐漸增加您的請求率。我們建議您以指數方式退避並再試一次。依預設，AWS 開發套件會使用自動重試邏輯和指數退避。如需詳細資訊，請參閱在 AWS 中錯誤重試與指數退避和指數退避和抖動。

OK to retry? (確定要重試嗎？) 是

VideoTooLargeException

訊息：影片大小 (位元組)：<影片大小> 超過最大限制：<最大大小> 位元組。

所提供媒體的檔案大小或持續時間太大。如需詳細資訊，請參閱Amazon Rekognition 中的準則和配額。

OK to retry? (確定要重試嗎？) 否

HTTP 狀態碼 5xx

HTTP 5xx 狀態碼指出 AWS 必須解決的問題。這可能是暫時性錯誤。如果確實如此，您可以重試請求直到成功。否則，請前往 AWS 服務運作狀態儀表板，確認服務是否發生任何操作問題。

InternalServerError(HTTP)

訊息：內部伺服器錯誤

Amazon Rekognition 發生服務問題。請再次嘗試呼叫。您應以指數方式退避，然後再試一次。依預設，AWS 開發套件會使用自動重試邏輯和指數退避。如需詳細資訊，請參閱在 AWS 中錯誤重試與指數退避和指數退避和抖動。

OK to retry? (確定要重試嗎？) 是

ThrottlingException(HTTP)

訊息：服務無法使用

Amazon Rekognition 暫時無法處理請求。請再次嘗試呼叫。我們建議您以指數方式退避並再試一次。依預設，AWS 開發套件會使用自動重試邏輯和指數退避。如需詳細資訊，請參閱在 AWS 中錯誤重試與指數退避和指數退避和抖動。

OK to retry? (確定要重試嗎？) 是

應用程式中的錯誤處理

若要讓您的應用程式順暢執行，您需要新增邏輯來截獲並回應錯誤。一般方式包含使用 `try-catch` 區塊或 `if-then` 陳述式。

AWS 開發套件會執行自己的重試和錯誤檢查。如果您在使用其中一個 AWS 開發套件時發生錯誤，則錯誤碼和說明可以協助您對其進行故障診斷。

您應該也會在回應中看到 `Request ID`。如果您需要與 AWS Support 合作來診斷問題，則 `Request ID` 可能十分有用。

以下 Java 程式碼片段嘗試偵測影像中的物件，並執行早期錯誤處理。(在此情況下，它只會通知請求失敗的使用者。)

```
try {
    DetectLabelsResult result = rekognitionClient.detectLabels(request);
    List <Label> labels = result.getLabels();

    System.out.println("Detected labels for " + photo);
    for (Label label: labels) {
        System.out.println(label.getName() + ": " + label.getConfidence().toString());
    }
}
catch(AmazonRekognitionException e) {
    System.err.println("Could not complete operation");
    System.err.println("Error Message:  " + e.getMessage());
    System.err.println("HTTP Status:    " + e.getStatusCode());
    System.err.println("AWS Error Code: " + e.getErrorCode());
    System.err.println("Error Type:     " + e.getErrorType());
    System.err.println("Request ID:     " + e.getRequestId());
}
catch (AmazonClientException ace) {
```

```
    System.err.println("Internal error occurred communicating with Rekognition");
    System.out.println("Error Message:  " + ace.getMessage());
}
```

在此程式碼片段中，try-catch 建構會處理兩個不同類型的例外狀況：

- AmazonRekognitionException— 如果用戶端請求已正確傳輸至 Amazon Rekognition，但 Amazon Rekognition 無法處理請求並傳回錯誤回應，就會發生此例外狀況。
- AmazonClientException— 如果用戶端無法從服務取得回應，或者用戶端無法剖析服務的回應，就會發生此例外狀況。

使用亞馬遜重新認知作為 FedRAMP 授權服務

該AWSFedRAMP 合規計劃包括亞馬遜重新認知作為 FedRAMP 授權的服務。如果您是聯邦或商業客戶，您可以在具有最高為中度影響層級資料的 AWS 美國東部和美國西部區域中，使用此服務來處理和存放機密工作負載。您可以將此服務用於AWS GovCloud（美國）地區的授權邊界，數據達到高影響力水平。如需 FedRAMP 合規的詳細資訊，請參閱 AWS FedRAMP 合規。

若要符合 FedRAMP 規範，您可以使用聯邦資訊處理標準 (FIPS) 端點。當您使用敏感資訊時，這可讓您存取 FIPS 140-2 驗證的密碼編譯模組。如需 FIPS 端點的相關詳細資訊，請參閱 FIPS 140-2 概觀。

您可以使用AWS Command Line Interface(AWS CLI) 或其中一個 AWS 開發套件，用來指定 Amazon 重新認知所使用的端點。

如需可與 Amazon 重新建立搭配使用的端點，請參閱亞馬遜重新認知區域和端點。

以下是列出集合主題中的亞馬遜重新認知開發人員指南。它們會經過修改，以指定可透過哪個區域和 FIPS 端點存取 Amazon Rekognition。

Java

如果是 Java，請使用withEndpointConfiguration當您構建亞馬遜重新認知客戶端的方法。此範例顯示您在美國東部 (維吉尼亞北部) 區域使用 FIPS 端點的集合：

```
//Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import java.util.List;
```

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.ListCollectionsRequest;
import com.amazonaws.services.rekognition.model.ListCollectionsResult;

public class ListCollections {

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


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


      System.out.println("Listing collections");
      int limit = 10;
      ListCollectionsResult listCollectionsResult = null;
      String paginationToken = null;
      do {
         if (listCollectionsResult != null) {
            paginationToken = listCollectionsResult.getNextToken();
         }
         ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
                 .withMaxResults(limit)
                 .withNextToken(paginationToken);

 listCollectionsResult=amazonRekognition.listCollections(listCollectionsRequest);

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

   }
```

```
}
```

AWS CLI

對於AWS CLI，使用--endpoint-url用於指定透過其存取 Amazon Rekognition 的端點的引數。下列範例顯示您在美國東部 (俄亥俄) 區域使用 FIPS 端點的集合：

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

Python

對於 Python，在 boto3.client 函數中使用 endpoint_url 引數。將其設定為您要指定的端點。此範例顯示您在美國西部 (奧勒岡) 區域使用 FIPS 端點的集合：

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

import boto3

def list_collections():

    max_results=2

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

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

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

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

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

        else:
            done=True

    return collection_count

def main():

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

感應器、輸入映像和影片的最佳實務

本節包含使用 Amazon Rekognition 的最佳實務資訊。

主題

Amazon Rekognition Image 操作延遲

為確保 Amazon Rekognition Image 操作的延遲盡可能降到最低，請考量下列事項：

- 含有您映像的 Amazon S3 儲存貯體區域必須符合您進行 Amazon Rekognition Image API 操作的區域。
- 使用映像位元組呼叫 Amazon Rekognition Image 操作，會比將映像上傳至 Amazon S3 儲存貯體，然後在 Amazon Rekognition Image 操作中參考已上傳映像更快。如果您想要將映像上傳至 Amazon Rekognition Image，進行近乎即時的處理，請考慮這個方法。例如，從 IP 相機上傳的映像或透過入口網站上傳的映像。
- 如果映像已在 Amazon S3 儲存貯體中，在 Amazon Rekognition Image 操作中對其進行參考可能會比將映像位元組傳遞至操作更快。

人臉比較輸入映像的建議

人臉比較操作所用的模型，是設計成適用各式各樣的姿勢、人臉表情、年齡範圍、旋轉、燈光條件以及大小。我們建議您在選擇參考相片時，使用以下準則將臉孔新增至系列IndexFaces。CompareFaces

用於人臉操作的輸入映像的一般建議

- 使用明亮且銳利的映像。盡量避免使用可能會因主體和相機操作而模糊的映像。DetectFaces可用於確定臉部的亮度和清晰度。

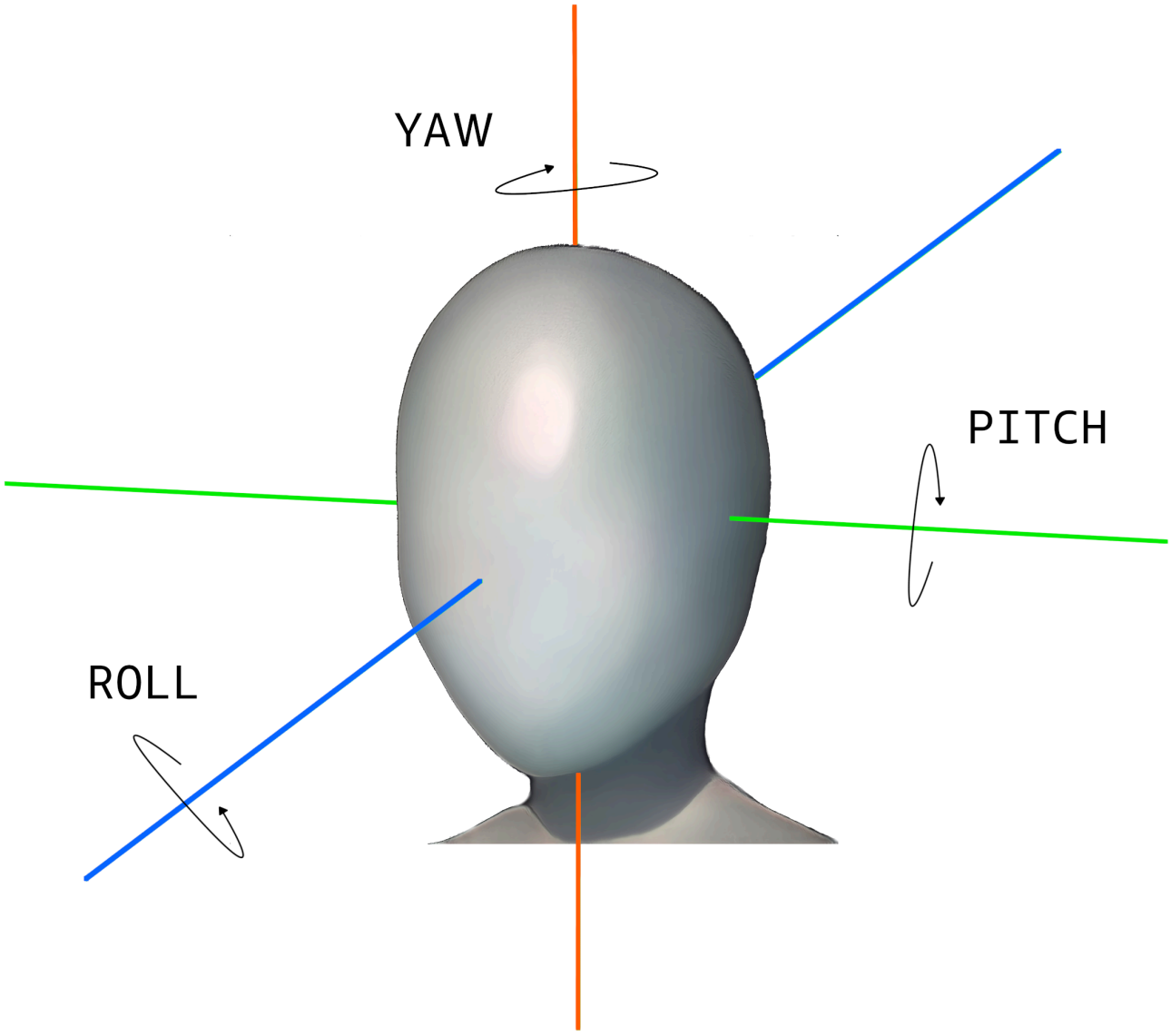
- 為了偵測視線，建議您以原始尺寸和質量上傳原始映像。
- 使用人臉在建議角度範圍內的映像。臉部俯角應該低於 30 度，而臉部仰角則低於 45 度。任一方向的擺角應該低於 45 度。翻滾沒有限制。
- 使用雙眼睜開且可見雙眼的人臉映像。
- 使用人臉沒有遭到隱蔽或緊密裁剪的映像。映像應包含肩膀以上的完整頭像。不應裁剪到臉部週框方塊。
- 避免擋住臉部的物品，如頭帶和口罩。
- 使用人臉佔據映像中大部分比例的映像。人臉佔據映像更大部分的映像比對時的準確性更高。
- 確保映像在分辨率方面足夠大。Amazon Rekognition 可以識別小至 50 x 50 像素的映像分辨率面孔，最高可達 1920 x 1080。映像的解析度愈高，需要的最小人臉大小愈大。大於大小下限的臉部提供一組更準確的臉部比較結果。
- 使用彩色映像。
- 使用人臉正面打光的映像，而不要有陰影等各種暗影。
- 使用與背景充分對比的映像。高對比單色背景的效果很好。
- 對於需要高精確度的應用，使用中性人臉表情的映像，嘴巴閉合，略帶或沒有笑容。

搜尋集合中人臉的建議

- 搜尋集合中的人臉時，請確保對最近的人臉映像建立索引。
- 使用 `IndexFaces` 建立集合時，請使用多個俯仰和偏擺不同的個人人臉映像 (在建議的角度範圍內)。我們建議一個人至少要有 5 個映像製成索引：正面、人臉向左偏擺 45 度或更少、人臉向右偏擺 45 度或更少、人臉俯角 30 度或更少、人臉仰角 45 度或更少。如果您想要追蹤這些人臉實例是否屬於同一個人，請在要製成索引的映像中只有一個人臉時，考慮使用外部映像 ID 屬性。例如，可在集合中利用外部映像 ID 作為 `John_Doe_1.jpg, … John_Doe_5.jpg` 來追蹤 John Doe 的 5 個映像。

相機設定的建議 (映像和影片)

除了人臉比較輸入映像的建議 之外，還有以下建議。



- 映像解析度：映像解析度沒有最低需求，只要人臉解析度為 50 x 50 像素，而映像解析度總計高達 1920 x 1080。映像的解析度愈高，需要的最小人臉大小愈大。

> **Note**
>
> 上述建議是根據相機的原生解析度。從低解析度映像產生高解析度映像不會產生人臉搜尋所需的結果 (因為成品是透過映像放大而產生的)。

- 相機角度：測量相機角度的方式有三種 (俯仰、翻滾和偏擺)。
  - 俯仰：我們建議當相機朝下時人臉俯角低於 30 度，朝上時人臉仰角低於 45 度。
  - 翻滾：此參數沒有最低要求。Amazon Rekognition 可以處理任何翻滾量。
  - 偏擺：我們建議任一方向的擺角低於 45 度。

  相機沿著任何軸擷取的臉部角度是面對場景的相機角度和主體頭部在場景中的角度的結合。例如，如果相機向下 30 度，而人員頭部又向下 30 度，則相機看到的實際臉部俯角為 60 度。在此情況下，Amazon Rekognition 無法辨識人臉。我們建議設定相機，以致相機角度根據下列假設：人們通常是以 30 度或更低的整體俯仰角 (臉部和相機的結合) 來看著相機。
- 相機變焦：建議的人臉解析度下限 50 x 50 像素應該驅動此相機設定。我們建議使用相機的變焦設定，以便所需臉部的解析度不低於 50 x 50 像素。
- 相機高度：建議的相機俯仰應該驅動此參數。

相機設定的建議 (已存放和串流影片)

除了 相機設定的建議 (映像和影片) 之外，還有以下建議。

- 編解碼器應該是 h.264 編碼。
- 建議的影格率為 30 fps。(它不得低於 5 fps。)
- 建議的編碼器位元速率為 3 Mbps。(它不得低於 1.5 Mbps。)
- 影格解析度：如果編碼器位元速率是一種限制，我們建議偏向使用更高的影格解析度，而非更高的影格率，以取得更好的人臉搜尋結果。這將確保 Amazon Rekognition 在設定的位元速率內取得最佳品質影格。不過，這有一個缺點。由於畫面播放速率較低，攝影機會遺失場景中的快速動作。對於特定的設定，了解這兩個參數之間的權衡是很重要的。例如，如果最大的可能位元速率為 1.5 Mbps，則相機可以 5 fps 擷取 1080p，或以 15 fps 擷取 720p。兩者之間的選擇取決於應用方式，只要符合建議的臉部解析度 50 x 50 像素即可。

相機設定的建議 (串流影片)

除了相機設定的建議 (已存放和串流影片) 之外，還有以下建議。

串流應用程式的額外限制是網際網路頻寬。若為即時影片，Amazon Rekognition 只接受 Amazon Kinesis Video Streams 做為輸入。您應該了解編碼器位元速率與可用網路頻寬之間的相依性。可用頻寬應該至少支援相機用來編碼即時串流的相同位元速率。這可確保將透過 Amazon Kinesis Video

Streams 轉送相機擷取的任何專案。如果可用頻寬低於編碼器位元速率，則 Amazon Kinesis Streams 就會根據網路頻寬捨棄位元。這會導致低影片品質。

一般串流設定涉及將多個相機連接到轉送串流的網路集線器。在此情況下，頻寬應該容納來自所有連接到集線器之相機的串流累計總和。例如，如果集線器連接到 5 部以 1.5 Mbps 編碼的相機，則可用的網路頻寬應該至少 7.5 Mbps。若要確保沒有捨棄的封包，您應該考慮將網路頻寬保持高於 7.5 Mbps，以因應由於相機與集線器之間捨棄的連線而產生的抖動。實際值將取決於內部網路的可靠性。

人臉活體的使用建議

使用人臉活體建議遵循下列最佳實務：

- 使用者應在不太暗或太亮且照明相當均勻的環境中完成人臉活體檢查。
- 使用者在瀏覽器上進行檢查時，應將其顯示屏的亮度提高到最高水平。行動裝置原生 SDK 會自動調整顯示器亮度。
- 選擇反映使用案例性質的可信度分數閾值。對於有更大安全考量的使用案例，請使用較高的閾值。
- 定期對稽核映像執行人工審核檢查，以確保在您設定的可信度閾值下降低詐騙攻擊。
- 如果您的使用者對照片敏感或不想使用 Rekognition 驗證其人臉活體，則為其提供替代的人臉活體驗證路徑。
- 請勿在使用者應用程式上傳送或顯示活體檢查分數。僅發送通過或失敗信號。
- 只允許在三分鐘內從單一裝置進行五次失敗的活體檢查。五次失敗後，將使用者逾時 30-60 分鐘。如果重複顯示 3 到 5 次模式，請阻止使用者裝置進行額外的呼叫。
- 在工作流程中實施準備就緒熒幕，以便使用者可以更輕鬆地通過人臉活體檢查。
- 您有責任向終端使用者提供合法適當的私隱通知，並獲得人臉活體處理、儲存、使用和轉移內容的必要同意。

偵測物件和概念

本節提供以 Amazon Rekognition Image 和 Amazon Rekognition Video 來偵測映像和影片內標籤的資訊。

標籤或標記是指映像中的物件或概念 (包括場景和動作)，根據映像或影片的內容決定。例如，熱帶海灘上的人物映像可能包含諸如棕櫚樹 (物件)、沙灘 (場景)、跑步 (動作) 和戶外 (概念) 等標籤。

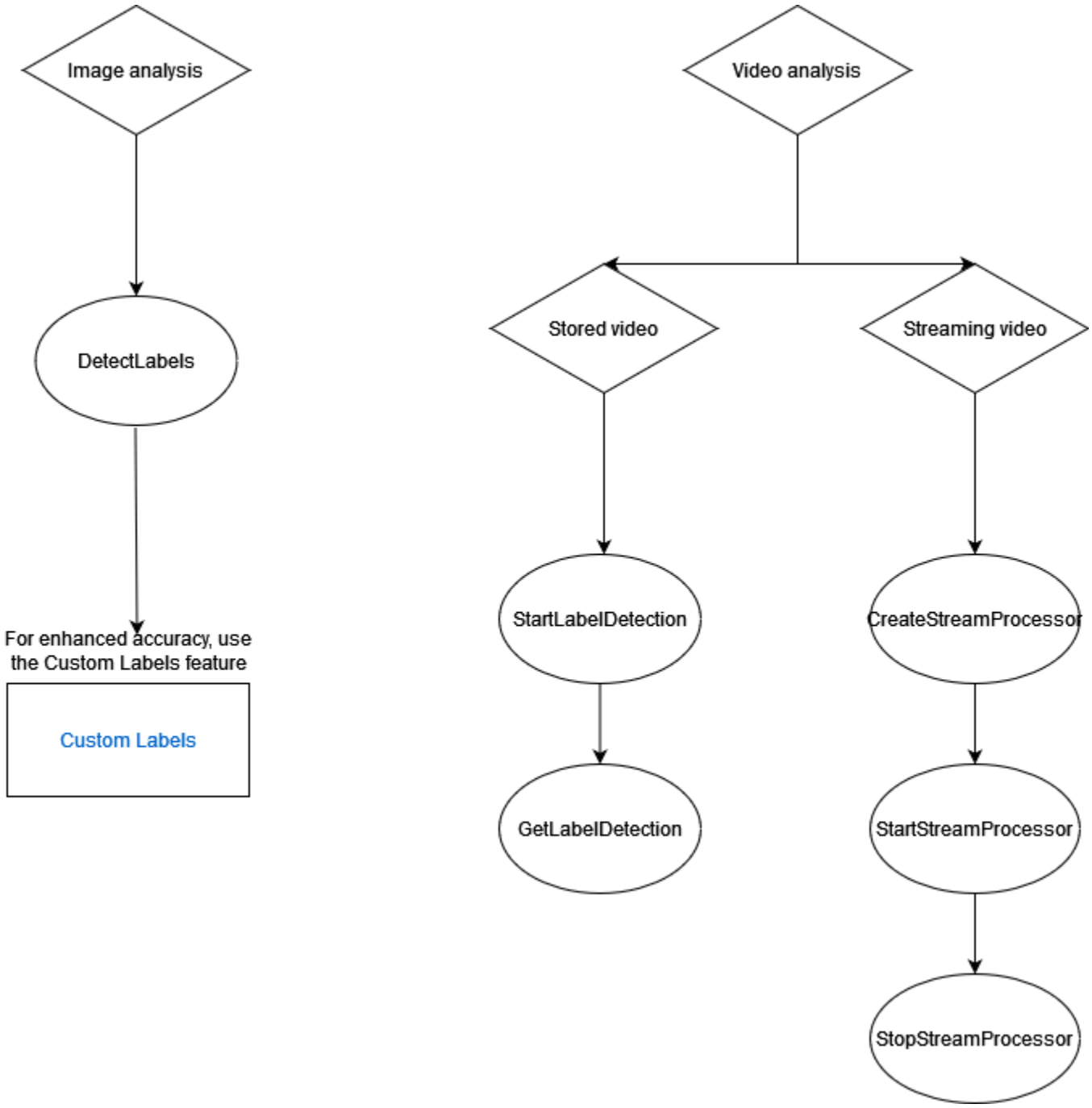
Rekognition 標籤偵測作業支援的標籤

- 若要下載 Amazon Rekognition 支援的最新標籤和物件邊界方塊清單，請按一下此處。
- 若要下載先前的標籤和物件邊界方框清單，請按一下此處。

> **Note**
>
> Amazon Rekognition 根據特定映像中人物的外觀進行性別二元制 (男人、女人、女孩等) 預測。這種預測不是為了對一個人的性別身份進行分類而設計，不應使用 Amazon Rekognition 做出該判定。比方說，可能將出于扮演角色需要戴假长发和耳環的男性演員預測為女性。
> 使用 Amazon Rekognition 進行性別二元論預測最適合用於需要分析彙總性別分佈統計資料而無需識別特定使用者的使用案例。例如，在社交媒體平台上，女性使用者與男性相比的百分比。
> 我們不建議採用性別二元論預測來制定會影響個人權利、隱私或服務存取的決策。

Amazon Rekognition 傳回英文標籤。您可以使用 Amazon Translate 將英文標籤翻譯成其他語言。

下圖顯示呼叫操作的順序，具體取決於您使用 Amazon Rekognition 影像或 Amazon Rekognition Video 操作的目標：



標籤回應物件

週框方塊

Amazon Rekognition Image 和 Amazon Rekognition Video 可以傳回常見物件標籤的週框方塊，例如人員、汽車、家具、服飾或寵物。對於較不常用的物件標籤，系統不會傳回週框方塊資訊。您可以使用

週框方塊，來找出物件在映像中的確切位置、已偵測到物件的計數實例，或使用週框方塊維度測量物件的大小。

例如，在以下映像中，Amazon Rekognition Image 能夠偵測人物、滑板、停好的車或其他資訊。Amazon Rekognition Image 也會傳回偵測到的人員的邊界外框，以及其他偵測到的物件 (例如汽車和車輪)。

可信度分數

Amazon Rekognition Video 和 Amazon Rekognition Image 提供了一個百分比分數，以表示 Amazon Rekognition 每個偵測到的標籤的準確性的可信度是多少。

父系

Amazon Rekognition Image 和 Amazon Rekognition Video 使用祖先標籤的階層分類法對標籤進行分類。例如，走在路上的人員可能會被偵測為行人。行人的父標籤為人員。在回應中會傳回這兩個標籤。系統會傳回所有上階標籤，而且指定的標籤包含一個清單，其中列出其父標籤和其他上階標籤。例如，

祖父和曾祖父標籤 (如果它們存在的話)。您可以使用父標籤來建置相關標籤的群組，並允許在一或多個映像中查詢類似標籤。例如，查詢所有車輛可能從某個映像傳回汽車，而從另一個映像傳回摩托車。

類別

Amazon Rekognition Image 和 Amazon Rekognition Video 傳回的有關標籤類別資訊。標籤是根據常用功能和環境 (例如「車輛和汽車」和「食品和飲料」) 將單個標籤組合在一起的類別的一部分。標示品類可以是父品類的子品類。

Aliases

除了傳回標籤之外，Amazon Rekognition Image 和 Amazon Rekognition Video 還會傳回與該標籤相關聯的任何別名。別名是具有相同含義的標籤或與傳回的主標籤可以在視覺上互換的標籤。例如，「手機」是「移動電話」的別名。

在舊版中，Amazon Rekognition Image 會在包含「手機」的主要標籤名稱清單中傳回「行動電話」等別名。Amazon Rekognition Image 目前在主標籤名稱清單中名為「別名」和「移動電話」的字段傳回「手機」。如果您的應用程式依賴於舊版 Rekognition 傳回的結構，您可能需要將映像或影片標籤偵測作業傳回的目前回應轉換為先前的回應結構，其中所有標籤和別名都會作為主要標籤傳回。

如果您需要將 DetectLabels API 的目前回應 (用於影像中的標籤偵測) 轉換為先前的回應結構，請參閱中的程式碼範例轉換響 DetectLabels應。

如果您需要將 GetLabelDetection API 的目前回應 (用於儲存視訊中的標籤偵測) 轉換為先前的回應結構，請參閱中的程式碼範例轉換響 GetLabelDetection 應。

映像屬性

Amazon Rekognition Image 會傳回整個映像的映像品質 (銳利度、亮度和對比度) 的相關資訊。映像的前景和背景也會傳回銳利度和亮度。「映像屬性」也可以用於偵測整個映像、前景、背景和具有邊界方框之物件的主要顏色。

以下是執行影像之 DetectLabels 作業回應中包含的 ImageProperties 資料範例：

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

映像屬性不適用於 Amazon Rekognition Video。

模型版本控制

Amazon Rekognition Image 和 Amazon Rekognition Video 都會傳回用來在映像或已儲存影片中偵測標籤的標籤偵測模型版本。

包含性或排斥性的篩選

您可以篩選 Amazon Rekognition Image 和 Amazon Rekognition Video 標籤偵測操作傳回的結果。透過提供標籤和類別的篩選條件來篩選結果。標籤篩選器可以是包含性或排斥性的。

請參閱 在映像中偵測標籤 有關由 `DetectLabels` 獲得的結果篩選的更多資訊。

請參閱 偵測影片中的標籤 有關由 `GetLabelDetection` 獲得的結果篩選的更多資訊。

排序和彙總結果

從某些 Amazon Rekognition Video 操作取得的結果可以根據時間戳記和影片區段進行排序和彙總。使用 GetLabelDetection 或 GetContentModeration 分別擷取標籤偵測或內容管制操作的結果時，您可以使用 SortBy 和 AggregateBy 引數來指定傳回結果的方式。您可以SortBy搭配TIMESTAMP或 NAME (標籤名稱) 使用，並使用TIMESTAMPS或SEGMENTS與 AggregateBy引數搭配使用。

在映像中偵測標籤

您可以使用此DetectLabels作業偵測影像中的標籤 (物件和概念)，並擷取有關影像屬性的資訊。映像屬性包括前景和背景顏色等屬性，以及映像的銳利度、亮度和對比度。您可以只檢索映像中的標籤，只檢索映像的屬性，或兩者兼而有之。如需範例，請參閱分析存放在 Amazon S3 儲存貯體中的映像。

下列範例使用各種 AWS SDK 和呼叫DetectLabels。 AWS CLI 如需有關 DetectLabels 操作回應的資訊，請參閱 DetectLabels 回應。

偵測映像中的標籤

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝並設定 AWS CLI 和 AWS SDK。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 將包含一個或多個物件的映像 (例如樹、房子和船) 上傳至您的 S3 儲存貯體。映像的格式必須是 .jpg 或 .png 格式。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例來呼叫 DetectLabels 操作。

   Java

   此範例顯示一份在輸入映像中偵測到的標籤清單。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。

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

```
import java.util.List;

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

public class DetectLabels {

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

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        DetectLabelsRequest request = new DetectLabelsRequest()
                .withImage(new Image().withS3Object(new
 S3Object().withName(photo).withBucket(bucket)))
                .withMaxLabels(10).withMinConfidence(75F);

        try {
            DetectLabelsResult result = rekognitionClient.detectLabels(request);
            List<Label> labels = result.getLabels();

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

                List<Instance> instances = label.getInstances();
                System.out.println("Instances of " + label.getName());
                if (instances.isEmpty()) {
                    System.out.println("  " + "None");
                } else {
```

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

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

AWS CLI

此範例顯示 detect-labels CLI 操作的 JSON 輸出。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。使用您開發人員設定檔的名稱取代 profile-name 的值。

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

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

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

Python

此範例顯示在輸入映像中偵測到的標籤。在函數 `main` 中，將 `bucket` 與 `photo` 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

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

import boto3

def detect_labels(photo, bucket):

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

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

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

```
        print("Instances:")

        for instance in label['Instances']:
            print(" Bounding box")
            print(" Top: " + str(instance['BoundingBox']['Top']))
            print(" Left: " + str(instance['BoundingBox']['Left']))
            print(" Width: " + str(instance['BoundingBox']['Width']))
            print(" Height: " + str(instance['BoundingBox']['Height']))
            print(" Confidence: " + str(instance['Confidence']))
            print()

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

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

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

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

     return len(response['Labels'])

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

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

.NET

此範例顯示一份在輸入映像中偵測到的標籤清單。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

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

        try
        {
```

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

Ruby

此範例顯示一份在輸入映像中偵測到的標籤清單。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。

```
  # Add to your Gemfile
  # gem 'aws-sdk-rekognition'
  require 'aws-sdk-rekognition'
  credentials = Aws::Credentials.new(
     ENV['AWS_ACCESS_KEY_ID'],
     ENV['AWS_SECRET_ACCESS_KEY']
  )
  bucket = 'bucket' # the bucket name without s3://
  photo  = 'photo' # the name of file
  client   = Aws::Rekognition::Client.new credentials: credentials
  attrs = {
    image: {
      s3_object: {
        bucket: bucket,
        name: photo
      },
    },
    max_labels: 10
  }
 response = client.detect_labels attrs
 puts "Detected labels for: #{photo}"
 response.labels.each do |label|
   puts "Label:      #{label.name}"
```

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

Node.js

此範例顯示一份在輸入映像中偵測到的標籤清單。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

如果您正在使用 TypeScript 定義，您可能需要使用import AWS from 'aws-sdk'而不是const AWS = require('aws-sdk')，以便使用 Node.js 運行該程序。您可以查閱適用於 Javascript 的AWS SDK，以獲取更多詳細資訊。視您設定組態的方式而定，您可能還需要使用 AWS.config.update({region:*region*}); 來指定您所在的區域。

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

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

const client = new AWS.Rekognition();
```

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

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import java.util.List;

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

    public static void main(String[] args) {

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

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

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
```

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

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

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    public static void getLabelsfromImage(RekognitionClient rekClient, String
 bucket, String image) {

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

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

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

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

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
// snippet-end:[rekognition.java2.detect_labels.main]
```

```
}
```

DetectLabels 操作請求

DetectLabel 的輸入是映像。在此範例 JSON 輸入中，來源映像是從 Amazon S3 儲存貯體載入的。MaxLabels 是回應中傳回的最大標籤數量。MinConfidence 是 Amazon Rekognition Image 在偵測到的標籤的準確性中必須具有的最低可信度，以便在回應中傳回。

此功能可讓您指定要傳回的映像的一或多個特徵，以便選取 GENERAL_LABELS 和 IMAGE_PROPERTIES。其包括的 GENERAL_LABELS 將傳回輸入映像中偵測到的標籤，IMAGE_PROPERTIES 將允許您存取映像的顏色和質量。

設定可讓您篩選 GENERAL_LABELS 和 IMAGE_PROPERTIES 特徵的傳回專案。對於標籤，您可以使用包容性和獨家篩選器。您還可以按特定標籤，單個標籤或按標籤類別進行篩選：

- LabelInclusionFilters -允許您指定要包含在響應中的標籤。
- LabelExclusionFilters -可讓您指定要從回應中排除的標籤。
- LabelCategoryInclusionFilters -可讓您指定要包含在回應中的標籤類別。
- LabelCategoryExclusionFilters -可讓您指定要從回應中排除的標籤類別。

您還可以根據需要組合包含性和排斥性篩選，但不包括某些標籤或類別以及包括其他標籤或類別。

IMAGE_PROPERTIES 指映像的主要色彩和品質屬性，例如銳利度、亮度和對比度。偵測 IMAGE_PROPERTIES 時，您可以使用 MaxDominantColors 參數指定要傳回的最大主色數 (預設值為 10)。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxLabels": 10,
    "MinConfidence": 75,
    "Features": [ "GENERAL_LABELS", "IMAGE_PROPERTIES" ],
    "Settings": {
```

```
        "GeneralLabels": {
            "LabelInclusionFilters": [<Label(s)>],
            "LabelExclusionFilters": [<Label(s)>],
            "LabelCategoryInclusionFilters": [<Category Name(s)>],
            "LabelCategoryExclusionFilters": [<Category Name(s)>]
        },
        "ImageProperties": {
            "MaxDominantColors":10
        }
    }
}
```

DetectLabels 回應

DetectLabels 的回應是映像中偵測到的一系列標籤，以及偵測所依據的可信度層級。

以下是 DetectLabels 的回應範例。以下範例回應包含針對 GENERAL_LABELS 傳回的各種屬性，包括：

- 名稱：偵測到的標籤的名稱。在此範例中，操作偵測到帶有標籤移動電話的物件。
- 可信度：每個標籤都有一個相關的可信度等級。在此範例中，標籤的可信度為 99.36%。
- 父項：偵測到之標籤的祖系標籤。在此範例中，標籤行動電話有一個名為電話的父標籤。
- 別名：標籤可能有別名的相關資訊。在此範例中，行動電話標籤有可能的行動電話別名。
- 類別：偵測到的標籤所屬的標籤類別。在這個例子中，其是技術和計算。

常見物件標籤的回應包含輸入映像上標籤位置的週框方塊資訊。例如，人員標籤具有一個實例陣列，其中包含兩個週框方塊。這些是在映像中偵測到的兩個人員位置。

回應還包括有關 IMAGE_PROPERTIES 的屬性。IMAGE_PROPERTIES 特徵所呈現的屬性如下：

- 質量：有關輸入映像的銳利度，亮度和對比度的資訊，得分在 0 到 100 之間。質量會針對整個映像以及映像的背景和前景報告品質 (如果有的話)。不過，只會報告整個映像的對比度，而銳利度和亮度也會用於報告背景和前景。
- 主色：映像中主色的陣列。每種主要顏色都使用簡化的顏色名稱、CSS 調色盤、RGB 值和十六進位程式碼來描述。
- 前景：有關輸入映像前景的主要顏色、銳利度和亮度的資訊。
- 背景：有關輸入映像背景的主要顏色、銳利度和亮度的資訊。

當 GENERAL_LABELS 和 IMAGE_PROPERTIES 一起使用做為輸入參數時，Amazon Rekognition Image 也會傳回具有邊界方塊之物件的主要顏色。

欄位 LabelModelVersion 包含 DetectLabels 所使用之偵測模型的版本編號。


```
{

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

```
                    }
                        ],
                }
            ]
        }
    ],
    "ImageProperties": {
        "Quality": {
            "Brightness": 40,
            "Sharpness": 40,
            "Contrast": 24,
        },
        "DominantColors": [
            {
                "Red": 120,
                "Green": 137,
                "Blue": 132,
                "HexCode": "3A7432",
                "SimplifiedColor": "red",
                "CssColor": "fuscia",
                "PixelPercentage": 40.10
            }
        ],
        "Foreground": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "red",
                    "PixelPercentage": 30.70
                }
            ],
        }
        "Background": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
```

```
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "Red",
                    "PixelPercentage": 10.20
                }
            ],
        },
    },
    "LabelModelVersion": "3.0"
}
```

轉換響 DetectLabels應

使用 DetectLabels API 時，您可能需要響應結構來模擬較舊的 API 響應結構，其中主要標籤和別名都包含在同一個列表中。

以下是目前 API 回應的範例 DetectLabels：

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

下面的例子顯示了來自 DetectLabelsAPI 的先前響應：

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

如果需要，您可以將當前回應轉換為遵循舊回應的格式。您可以使用下列範例程式碼，將最新的 API 回應轉換為先前的 API 回應結構：

Python

下列程式碼範例示範如何從 DetectLabels API 轉換目前的回應。在下面的代碼示例中，您可以將 *EXAMPLE_INFERENCE_OUTP* UT 的值替換為您已經運行的操作的輸出。 DetectLabels

```
from copy import deepcopy

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

#Latest API response sample
EXAMPLE_INFERENCE_OUTPUT = {
    "Labels": [
```

```
        {
            "Name": "Mobile Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
            ],
            "Aliases": [
                {
                    "Name": "Cell Phone"
                }
            ],
            "Instances":[
                {
                    "BoundingBox":{
                        "Height":0.1549897,
                        "Width":0.07747964,
                        "Top":0.50858885,
                        "Left":0.00018205095
                    },
                    "Confidence":98.401276
                }
            ]
        },
        {
            "Name": "Urban",
            "Confidence": 99.99982,
            "Categories": [
                "Colors and Visual Composition"
            ]
        }
    ]
}

def expand_aliases(inferenceOutputsWithAliases):

    if LABEL_KEY in inferenceOutputsWithAliases:
        expandInferenceOutputs = []
        for primaryLabelDict in inferenceOutputsWithAliases[LABEL_KEY]:
            if ALIASES_KEY in primaryLabelDict:
                for alias in primaryLabelDict[ALIASES_KEY]:
                    aliasLabelDict = deepcopy(primaryLabelDict)
                    aliasLabelDict[NAME_KEY] = alias[NAME_KEY]
```

```
                    del aliasLabelDict[ALIASES_KEY]
                    if INSTANCE_KEY in aliasLabelDict:
                        del aliasLabelDict[INSTANCE_KEY]
                    expandInferenceOutputs.append(aliasLabelDict)

        inferenceOutputsWithAliases[LABEL_KEY].extend(expandInferenceOutputs)

    return inferenceOutputsWithAliases


if __name__ == "__main__":

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

以下是轉換回應的範例：

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

```
                }
            ]
        },
        {
            "Name": "Cell Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
            ],
            "Instances":[]
        },
        {
            "Name": "Urban",
            "Confidence": 99.99982,
            "Categories": [
                "Colors and Visual Composition"
            ]
        }
    ]
}
```

偵測影片中的標籤

Amazon Rekognition Video 可以在影片中偵測標籤 (物件和概念)，以及偵測到標籤的時間。如需開發套件程式碼範例，請參閱 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。如需範 AWS CLI 例，請參閱使用分析視訊 AWS Command Line Interface。

Amazon Rekognition Video 標籤偵測是一種非同步操作。若要開始偵測視訊中的標籤，請呼叫StartLabel偵測。

Amazon Rekognition Video 向 Amazon Simple Notification Service 主題發佈影片的完成狀態。如果視頻分析成功，請調用「檢測」以獲取GetLabel檢測到的標籤。如需呼叫影片分析 API 操作的資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

StartLabel檢測請求

以下是 StartLabelDetection 操作要求的範例。您可以使用儲存在 Amazon S3 儲存貯體中的影片來提供 StartLabelDetection 操作。在範例請求 JSON 中，會指定 Amazon S3 儲存貯體和影片名稱以及 MinConfidence、Features、Settings 和 NotificationChannel。

MinConfidence 是 Amazon Rekognition Video 在偵測到的標籤的準確性中必須具有的最低可信度或執行個體的週框方塊 (如偵測到)，以便在回應中傳回。

透過 Features，您可以指定要將 GENERAL_LABELS 作為回應的一部分傳回。

使用 Settings，您可以篩選 GENERAL_LABELS 傳回的專案。對於標籤，您可以使用包容性和獨家篩選器。您還可以按特定標籤，單個標籤或按標籤類別進行篩選：

- LabelInclusionFilters：用於指定要包含在回應中的標籤
- LabelExclusionFilters：用於指定要從回應中排除的標籤。
- LabelCategoryInclusionFilters：用於指定要包含在回應中的標籤類別。
- LabelCategoryExclusionFilters：用於指定要從回應中排除的標籤類別。

您還可以根據需要組合包含性和排斥性篩選，但不包括某些標籤或類別以及包括其他標籤或類別。

NotificationChannel 是您希望 Amazon Rekognition Video 將標籤偵測操作的完成狀態發佈到 Amazon SNS 主題的 ARN。如果您使用的是 AmazonRekognitionServiceRole 許可政策，則 Amazon SNS 主題必須具有以 Rekognition 開頭的主題名稱。

以下是 JSON 格式的範例 StartLabelDetection 要求，包括篩選條件：

```
{
    "ClientRequestToken": "5a6e690e-c750-460a-9d59-c992e0ec8638",
    "JobTag": "5a6e690e-c750-460a-9d59-c992e0ec8638",
    "Video": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "video.mp4"
         }
     },
     "Features": ["GENERAL_LABELS"],
     "MinConfidence": 75,
     "Settings": {
         "GeneralLabels": {
             "LabelInclusionFilters": ["Cat", "Dog"],
```

```
            "LabelExclusionFilters": ["Tiger"],
            "LabelCategoryInclusionFilters": ["Animals and Pets"],
            "LabelCategoryExclusionFilters": ["Popular Landmark"]
        }
    },
    "NotificationChannel": {
        "RoleArn": "arn:aws:iam::012345678910:role/SNSAccessRole",
        "SNSTopicArn": "arn:aws:sns:us-east-1:012345678910:notification-topic",
    }
}
```

GetLabelDetection 作業回應

GetLabelDetection 會傳回陣列 (Labels)，其中包含影片中偵測到之標籤的相關資訊。陣列可依時間或指定 SortBy 參數時偵測到的標籤來排序。您也可以使用 AggregateBy 參數來選取回應專案的彙總方式。

以下是 GetLabelDetection 的 JSON 回應範例。在回應中，請注意下列事項：

- 排序順序：傳回的標籤陣列會依時間排序。若要依標籤排序，請在 SortBy 輸入參數中指定 NAME 以執行 GetLabelDetection。如果標籤在視頻中出現多次，則會出現（LabelDetection）元素的倍數實例。預設排序或是 TIMESTAMP，而次要排序順序為 NAME。
- 標籤資訊：LabelDetection 陣列元素包含 (標籤) 物件，其中包括標籤名稱和 Amazon Rekognition 標籤偵測精確度的可信度分數。Label 物件也包含標籤的階層式分類法和常用標籤的週框方塊資訊。Timestamp 則是偵測到標籤的時間，從影片開始起算並以毫秒為單位。

  也會傳回與標籤相關聯之任何類別或別名的相關資訊。對於依影片 SEGMENTS 彙總的結果，會傳回 StartTimestampMillis、EndTimestampMillis 和 DurationMillis 的結構，分別定義區段的開始時間、結束時間和持續時間。
- 彙總：指定傳回時如何彙總結果。依據 TIMESTAMPS 彙總預設值。您也可以選擇依據 SEGMENTS 彙總，以便在時間範圍內彙總結果。如果依據 SEGMENTS 彙總，則不會傳回偵測到具有邊界方框之執行個體的相關資訊。僅傳回區段期間偵測到的標籤。
- 分頁資訊：範例顯示標籤偵測資訊的一頁。您可以指定 GetLabelDetection 的 MaxResults 輸入參數中要傳回幾個 LabelDetection 物件。如果結果數目超過 MaxResults，GetLabelDetection 會傳回用來取得下一頁結果的字符 (NextToken)。如需詳細資訊，請參閱 取得 Amazon Rekognition Video 分析結果。
- 影片資訊：回應包含 GetLabelDetection 所傳回之每頁資訊中影片格式 (VideoMetadata) 的相關資訊。

以下是 JSON 格式的範例 GetLabelDetection 回應，其中包含時間戳記彙總：

```
{
    "JobStatus": "SUCCEEDED",
    "LabelModelVersion": "3.0",
    "Labels": [
        {
            "Timestamp": 1000,
            "Label": {
                "Name": "Car",
                "Categories": [
                  {
                    "Name": "Vehicles and Automotive"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Automobile"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875, // Classification confidence
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567
                        },
                        "Confidence": 99.9364013671875 // Detection confidence
                    }
                ]
            }
        },
        {
            "Timestamp": 1000,
            "Label": {
                "Name": "Cup",
                "Categories": [
```

```
                {
                    "Name": "Kitchen and Dining"
                }
            ],
            "Aliases": [
                {
                    "Name": "Mug"
                }
            ],
            "Parents": [],
            "Confidence": 99.9364013671875, // Classification confidence
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.26779675483703613,
                        "Height": 0.8562285900115967,
                        "Left": 0.3604024350643158,
                        "Top": 0.09245597571134567
                    },
                    "Confidence": 99.9364013671875 // Detection confidence
                }
            ]
        }
    },
    {
        "Timestamp": 2000,
        "Label": {
            "Name": "Kangaroo",
            "Categories": [
                {
                    "Name": "Animals and Pets"
                }
            ],
            "Aliases": [
                {
                    "Name": "Wallaby"
                }
            ],
            "Parents": [
                {
                    "Name": "Mammal"
                }
            ],
            "Confidence": 99.9364013671875,
```

```
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567,
                        },
                        "Confidence": 99.9364013671875
                    }
                ]
            }
        },
        {
            "Timestamp": 4000,
            "Label": {
                "Name": "Bicycle",
                "Categories": [
                  {
                    "Name": "Hobbies and Interests"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Bike"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567
                        },
                        "Confidence": 99.9364013671875
                    }
                ]
```

```
            }
        }
    ],
    "VideoMetadata": {
        "ColorRange": "FULL",
        "DurationMillis": 5000,
        "Format": "MP4",
        "FrameWidth": 1280,
        "FrameHeight": 720,
        "FrameRate": 24
    }
}
```

以下是 JSON 格式的範例 GetLabelDetection 回應，其中包含按區段彙總：

```
{
    "JobStatus": "SUCCEEDED",
    "LabelModelVersion": "3.0",
    "Labels": [
        {
            "StartTimestampMillis": 225,
            "EndTimestampMillis": 3578,
            "DurationMillis": 3353,
            "Label": {
                "Name": "Car",
                "Categories": [
                  {
                    "Name": "Vehicles and Automotive"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Automobile"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875 // Maximum confidence score for Segment
 mode
            }
```

```
        },
        {
            "StartTimestampMillis": 7578,
            "EndTimestampMillis": 12371,
            "DurationMillis": 4793,
            "Label": {
                "Name": "Kangaroo",
                "Categories": [
                  {
                    "Name": "Animals and Pets"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Wallaby"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Mammal"
                  }
                ],
                "Confidence": 99.9364013671875
            }
        },
        {
            "StartTimestampMillis": 22225,
            "EndTimestampMillis": 22578,
            "DurationMillis": 2353,
            "Label": {
                "Name": "Bicycle",
                "Categories": [
                  {
                    "Name": "Hobbies and Interests"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Bike"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
```

```
                }
            ],
            "Confidence": 99.9364013671875
        }
    }
],
"VideoMetadata": {
    "ColorRange": "FULL",
    "DurationMillis": 5000,
    "Format": "MP4",
    "FrameWidth": 1280,
    "FrameHeight": 720,
    "FrameRate": 24
}
}
```

轉換響 GetLabelDetection 應

使用 GetLabelDetection API 作業擷取結果時，您可能需要回應結構來模擬較舊的 API 回應結構，其中主要標籤和別名都包含在相同的清單中。

上一節中找到的 JSON 回應範例會顯示 API 回應的目前格式 GetLabelDetection。

下面的例子顯示了來自 GetLabelDetection API 的先前響應：

```
{
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 60.51791763305664,
                "Parents": [],
                "Name": "Leaf"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Human"
```

```
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.11109819263219833,
                            "Top": 0.08098889887332916,
                            "Left": 0.8881205320358276,
                            "Height": 0.9073750972747803
                        },
                        "Confidence": 99.5831298828125
                    },
                    {
                        "BoundingBox": {
                            "Width": 0.1268676072359085,
                            "Top": 0.14018426835536957,
                            "Left": 0.0003282368124928324,
                            "Height": 0.7993982434272766
                        },
                        "Confidence": 99.46029663085938
                    }
                ],
                "Confidence": 99.63411102294922,
                "Parents": [],
                "Name": "Person"
            }
        },
        .
        .
        .

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

```
                "Name": "Sleeve"
            }
        }

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

如果需要，您可以將當前回應轉換為遵循舊回應的格式。您可以使用下列範例程式碼，將最新的 API 回應轉換為先前的 API 回應結構：

```
from copy import deepcopy

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

#Latest API response sample for AggregatedBy SEGMENTS
EXAMPLE_SEGMENT_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Person",
                "Confidence": 97.530106,
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
```

```
                    {
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],

            },
            "StartTimestampMillis": 6400,
            "EndTimestampMillis": 8200,
            "DurationMillis": 1800
        },
    ]
}

#Output example after the transformation for AggregatedBy SEGMENTS
EXPECTED_EXPANDED_SEGMENT_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Person",
                "Confidence": 97.530106,
                "Parents": [],
                "Aliases": [
```

```
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],

            },
            "StartTimestampMillis": 6400,
            "EndTimestampMillis": 8200,
            "DurationMillis": 1800
        },
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Human",
                "Confidence": 97.530106,
                "Parents": [],
                "Categories": [
                    {
```

```
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
    ]
}

#Latest API response sample for AggregatedBy TIMESTAMPS
EXAMPLE_TIMESTAMP_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Person",
                "Confidence": 97.530106,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Height": 0.1549897,
                            "Width": 0.07747964,
                            "Top": 0.50858885,
                            "Left": 0.00018205095
                        },
                        "Confidence": 97.530106
                    },
                ],
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
        {
```

```
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Instances": [],
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],
            },
        },
    ]
}

#Output example after the transformation for AggregatedBy TIMESTAMPS
EXPECTED_EXPANDED_TIMESTAMP_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Person",
                "Confidence": 97.530106,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Height": 0.1549897,
                            "Width": 0.07747964,
                            "Top": 0.50858885,
                            "Left": 0.00018205095
                        },
                        "Confidence": 97.530106
                    },
                ],
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
```

```
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Instances": [],
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],
            },
        },
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Human",
                "Confidence": 97.530106,
                "Parents": [],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
    ]
}
```

```
def expand_aliases(inferenceOutputsWithAliases):

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

        inferenceOutputsWithAliases[VIDEO_LABEL_KEY].extend(expandInferenceOutputs)

    return inferenceOutputsWithAliases


if __name__ == "__main__":

    segmentOutputWithExpandAliases = expand_aliases(EXAMPLE_SEGMENT_OUTPUT)
    assert segmentOutputWithExpandAliases == EXPECTED_EXPANDED_SEGMENT_OUTPUT

    timestampOutputWithExpandAliases = expand_aliases(EXAMPLE_TIMESTAMP_OUTPUT)
    assert timestampOutputWithExpandAliases == EXPECTED_EXPANDED_TIMESTAMP_OUTPUT
```

偵測串流視訊事件中的標籤

您可以使用亞馬遜視訊偵測串流視訊中的標籤。要做到這一點，你創建一個流處理器（ CreateStreamProcessor) 開始和管理串流視訊的分析。

亞馬遜重新認知影片使用 Amazon Kinesis 影片串流來接收和處理視訊串流。建立串流處理器時，您可以選擇要串流處理器偵測的項目。您可以選擇人，包和寵物，或人和套餐。分析結果會輸出到您的 Amazon S3 儲存貯體和 Amazon SNS 通知中。請注意，Amazon Rekognition Video 會偵測到影片中有人的存在，但不會偵測該人員是否為特定個體。若要在串流影片中搜尋系列中的臉孔，請參閱the section called “在串流影片中搜尋集合中的人臉”。

若要將 Amazon Rekognition 視訊與串流視訊搭配使用，您的應用程式需要下列條件：

- Kinesis 視訊串流，用於將串流視訊傳送至亞馬遜 Rekognition 視訊。如需詳細資訊，請參閱亞馬遜 Kinesis 影片串流開發人員指南。
- 用於管理串流影片分析的 Amazon Rekognition 視訊串流處理器。如需詳細資訊，請參閱亞馬遜視訊串流處理器操作概觀。
- Amazon S3 儲存貯體。亞馬遜重新認知影片會將工作階段輸出發佈到 S3 儲存貯體。輸出包括第一次偵測到感興趣的人物或物體的影像框。您必須是 S3 儲存貯體的擁有者。
- 亞馬遜認知影片發佈智慧警示和電子郵件的 Amazon SNS 主題nd-of-session摘要至。

主題

設定您的亞馬遜認知影片和亞馬遜 Kinesis 資源

下列程序說明佈建 Kinesis 視訊串流所採取的步驟，以及用來偵測串流視訊中標籤的其他資源。

先決條件

若要執行此程序，AWS SDK for Java必須安裝。如需詳細資訊，請參閱Amazon Rekognition 入門。該AWS您使用的帳戶需要 Amazon Rekognition API 的存取權限。如需詳細資訊，請參閱〈亞馬遜重新認知所定義的動作在IAM 使用者指南。

偵測影片串流中的標籤 (AWS 開發套件)

1. 建立 Amazon S3 儲存貯體。請記下值區名稱以及您要使用的任何金鑰前置詞。您稍後會使用此資訊。
2. 建立一個 Amazon SNS 主題。您可以使用它在視頻流中首次檢測到感興趣的對象時接收通知。請注意該主題的亞馬遜資源名稱（ARN）。如需詳細資訊，請參閱〈創建一個亞馬遜 SNS 主題在亞馬遜 SNS 開發人員指南中。
3. 訂閱亞馬遜 SNS 主題的端點。如需詳細資訊，請參閱〈訂閱亞馬遜 SNS 主題在亞馬遜 SNS 開發人員指南中。
4. 建立室壁運動影片串流並記下流的亞馬遜資源名稱（ARN）。
5. 如果您尚未建立 IAM 服務角色，請建立 IAM 服務角色，讓 Amazon Rekognition 影片存取權存取您的 Kinesis 影片串流、S3 儲存貯體和 Amazon SNS 主題。如需詳細資訊，請參閱提供標籤檢測流處理器的訪問權限。

然後你可以創建標籤檢測流處理器和啟動串流處理器使用您選擇的串流處理器名稱。

> **Note**
>
> 只有在確認可以將媒體內嵌到 Kinesis 視訊串流後，才啟動串流處理器。

攝影機方向和設定

Amazon Rekognition 視訊串流影片事件可支援 Kinesis 影片串流支援的所有攝影機。為了獲得最佳效果，我們建議將相機放置在距離地面 0 到 45 度之間。相機必須保持標準的直立位置。例如，如果框架中有一個人，那個人應該垂直定位，人的頭部在框架中應該高於腳。

提供標籤檢測流處理器的訪問權限

您使用AWS Identity and Access Management(IAM) 服務角色，可授予 Amazon Rekognition 影片讀取 Kinesis 影片串流的存取權限。若要這麼做，請使用 IAM 角色將 Amazon Rekognition 影片存取權授予 Amazon S3 儲存貯體和 Amazon SNS 主題的存取權。

您可以建立允許 Amazon Rekognition 影片存取現有 Amazon SNS 主題、Amazon S3 儲存貯體和 Kinesis 影片串流的許可政策。對於一個step-by-step程序使用AWS CLI，請參閱the section called “AWS CLI用於設定標籤偵測 IAM 角色的命令”。

授予亞馬遜視頻訪問用於標籤檢測的資源

1. 使用 IAM JSON 政策編輯器建立新的許可政策，並使用下列原則。取代`kvs-stream-name`與室壁運動視頻流的名稱，`topicarn`使用您要使用的亞馬遜 SNS 主題的亞馬遜資源名稱（ARN），以及`bucket-name`與亞馬遜 S3 存儲桶的名稱。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KinesisVideoPermissions",
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
```

```
            "Resource": [
                "arn:aws:kinesisvideo:::stream/kvs-stream-name/*"
            ]
        },
        {
            "Sid": "SNSPermissions",
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:::sns-topic-name"
            ]
        },
        {
            "Sid": "S3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

2. 建立 IAM 服務角色，或更新現有的 IAM 服務角色。使用下列資訊建立 IAM 服務角色：

   1. 針對服務名稱選擇 Rekognition。
   2. 為服務角色使用案例選擇 Rekognition。
   3. 連接您在步驟 1 中建立的許可政策。

3. 請記下服務角色 ARN。在執行視頻分析操作之前，您需要它來創建流處理器。
4. （可選）如果您使用自己的AWS KMS要加密傳送到 S3 儲存貯體的資料的金鑰，您必須使用 IAM 角色新增以下陳述式。這是您為金鑰政策建立的 IAM 角色，對應於您要使用的客戶管理金鑰。）

```
            {
                        "Sid": "Allow use of the key by label detection Role",
                        "Effect": "Allow",
                        "Principal": {
```

```
                            "AWS":
 "arn:aws:iam:::role/REPLACE_WITH_LABEL_DETECTION_ROLE_CREATED"
                        },
                        "Action": [
                            "kms:Decrypt",
                            "kms:GenerateDataKey*"
                        ],
                        "Resource": "*"
            }
```

AWS CLI用於設定標籤偵測 IAM 角色的命令

如果您還沒有，請設置和配置AWS CLI用你的憑據。

將下列命令輸入AWS CLI設定具有標籤偵測所需權限的 IAM 角色。

1. `export IAM_ROLE_NAME=labels-test-role`
2. `export AWS_REGION=us-east-1`
3. 建立信任關係原則檔案 (例如，assume-role-rekognition.json) 具有以下內容。

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

4. `aws iam create-role --role-name $IAM_ROLE_NAME --assume-role-policy-document file://path-to-assume-role-rekognition.json --region $AWS_REGION`

5. `aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn "arn:aws:iam::aws:policy/service-role/AmazonRekognitionServiceRole" --region $AWS_REGION`
6. 如果您要接收通知的 SNS 主題名稱不是以」開頭AmazonRekognition「前綴，添加以下策略：

   `aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn "arn:aws:iam::aws:policy/AmazonSNSFullAccess" --region $AWS_REGION`
7. 如果您使用自己的 AWS KMS 金鑰加密傳送到 Amazon S3 儲存貯體的資料，請更新您要使用之客戶受管金鑰的金鑰政策。
   a. 建立一個包含下列內容的檔案：

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

   b. `export KMS_KEY_ID=labels-kms-key-id`。 將 KMS_KEY_ID 取代為您建立的 KMS 金鑰識別碼。
   c. `aws kms put-key-policy --policy-name default --key-id $KMS_KEY_ID --policy file://path-to-kms-key-policy.json`

串流視訊事件的標籤偵測作業

Amazon Rekognition Video 可以偵測串流影片中的人員或相關物件，並在偵測到它們時通知您。建立標籤偵測串流處理器時，請選擇您希望 Amazon Rekognition 視訊偵測的標籤。這些可以是人，包和寵物，或人，包和寵物。僅選擇要檢測的特定標籤。這樣，唯一相關的標籤就會創建通知。您可以設定選項以決定何時儲存視訊資訊，然後根據影格中偵測到的標籤執行其他處理。

設定資源後，在串流視訊中偵測標籤的程序如下：

1. 建立串流處理器
2. 啟動串流處理器
3. 如果偵測到感興趣的物件，您會在每個感興趣的物件第一次出現時收到 Amazon SNS 通知。
4. 當指定的時間時，流處理器停止`MaxDurationInSeconds`已完成。
5. 您會收到最終的 Amazon SNS 通知，其中包含事件摘要。
6. 亞馬遜重新認知影片會將詳細的工作階段摘要發佈到您的 S3 儲存貯體。

主題

建立 Amazon 重新認知視訊標籤偵測串流處理器

在您可以分析串流視訊之前，請先建立 Amazon Rekognition 視訊串流處理器 (CreateStreamProcessor).

如果您想要建立串流處理器來偵測感興趣的標籤和人員，請提供 Kinesis 視訊串流作為輸入 (`Input`），亞馬遜 S3 存儲桶信息（`Output`）和一個亞馬遜 SNS 主題 ARN（`StreamProcessorNotificationChannel`). 您也可以提供 KMS 金鑰識別碼，以加密傳送至 S3 儲存貯體的資料。您指定要在其中檢測的內容`Settings`，例如人員，包裹和人，或寵物，人和包裹。您也可以在框架中指定您希望 Amazon Rekognition 監控的位置`RegionsOfInterest`。以下是 `CreateStreamProcessor` 要求的 JSON 範例。

```
{
  "DataSharingPreference": { "OptIn":TRUE
  },
  "Input": {
    "KinesisVideoStream": {
      "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnnn:stream/muh_video_stream/
nnnnnnnnnnnn"
    }
  },
  "KmsKeyId": "muhkey",
```

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

```
  ]
}
```

請注意，您可以變更MinConfidence值，當您指定ConnectedHomeSettings用於流處理器。MinConfidence是一個介於 0 到 100 之間的數值，表示演算法對其預測的確定性。例如，通知person90 的置信度值意味著該算法絕對可以肯定的是，該人存在於視頻中。信心值為 10 表示可能有一個人。您可以設置MinConfidence在 0 到 100 之間選擇的所需值，具體取決於您希望收到通知的頻率。例如，如果您只想在 Rekognition 絕對確定視頻幀中有包時收到通知，則可以設置MinConfidence到九十年。

默認情況下，MinConfidence 設定為 50。如果要優化算法以獲得更高的精度，則可以設置MinConfidence要高於 50。然後，您收到的通知較少，但每個通知都更可靠。如果你想優化算法以獲得更高的調用，那麼你可以設置MinConfidence低於 50 以接收更多通知。

啟動亞馬遜視頻標籤檢測流處理器

您可以透過呼叫 StartStreamProcessor 並提供您在 CreateStreamProcessor 中指定的串流處理器名稱，來開始分析串流影片。當您執行StartStreamProcessor在標籤檢測流處理器上操作，您可以輸入啟動和停止信息以確定處理時間。

啟動串流處理器時，標籤偵測串流處理器狀態會以下列方式變更：

1. 當你打電話StartStreamProcessor，標籤檢測流處理器狀態從STOPPED或者FAILED至STARTING。
2. 在標籤檢測流處理器運行時，它會保持在STARTING。
3. 當標籤檢測流處理器完成運行時，狀態變為STOPPED或者FAILED。

該StartSelector指定 Kinesis 串流中要開始處理的起點。您可以使用 KVS 生產者時間戳記或 KVS 片段編號。如需詳細資訊，請參閱〈片段。

> **Note**
>
> 如果您使用 KVS 生產者時間戳記，則必須輸入以毫秒為單位的時間。

該StopSelector指定何時停止處理流。您可以指定處理視訊的時間上限。預設值為 10 秒的最長持續時間。請注意，實際處理時間可能會比最大持續時間長一點，具體取決於個別 KVS 片段的大小。如果已達到或超過片段末尾的持續時間上限，處理時間就會停止。

以下是 StartStreamProcessor 要求的 JSON 範例。

```
{
   "Name": "string",
   "StartSelector": {
     "KVSStreamStartSelector": {
         "KVSProducerTimestamp": 1655930623123
      },
        "StopSelector": {
            "MaxDurationInSeconds": 11
      }
   }
}
```

如果串流處理器成功啟動，則會傳回 HTTP 200 回應。此時會包含一個空的 JSON 主體。

分析標籤偵測結果

Amazon Rekognition 影片從標籤偵測串流處理器發佈通知的方式有三種：用於物件偵測事件的 Amazon SNS 通知、電子郵件的 Amazon SNS 通知nd-of-session摘要，以及詳細的 Amazon S3 儲存貯體報告。

- 針對物件偵測事件的 Amazon SNS 通知。

  如果在視訊串流中偵測到標籤，您會收到有關物件偵測事件的 Amazon SNS 通知。Amazon Rekognition 會在視訊串流中第一次偵測到人員或感興趣的物件時發佈通知。通知包括偵測到的標籤類型、可信度，以及英雄影像的連結等資訊。它們還包括檢測到的人物或對象的裁剪圖像以及檢測時間戳。通知的格式如下：

  ```
  {"Subject": "Rekognition Stream Processing Event",
      "Message": {
          "inputInformation": {
              "kinesisVideo": {
                  "streamArn": string
              }
  ```

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

- 亞馬遜 SNS end-of-session摘要。

  當串流處理工作階段完成時，您也會收到 Amazon SNS 通知。此通知會列出工作階段的中繼資料。這包括詳細資料，例如處理的串流持續時間。通知的格式如下：

```
{"Subject": "Rekognition Stream Processing Event",
    "Message": {
        "inputInformation": {
```

```
            "kinesisVideo": {
                "streamArn": string,
                "processedVideoDurationMillis": number
            }
        },
        "eventNamespace": {
            "type": "STREAM_PROCESSING_COMPLETE"
        },
        "streamProcessingResults": {
            "message": string
        },
        "eventId": string,
        "tags": {
            [string]: string
        },
        "sessionId": string,
        "startStreamProcessorRequest": object
    }
}
```

- 亞馬遜 S3 存儲桶報告。

  亞馬遜 Rekognition 影片會將影片分析操作的詳細推論結果發佈到亞馬遜 S3 儲存貯體中提供的`CreateStreamProcessor`操作。這些結果包括第一次偵測到感興趣的物件或人物的影像框。

  S3 中的框架在以下路徑中可用：ObjectKeyPrefix/StreamProcessorName/SessionId/########。在這條路上，LabelKeyPrefix是客戶提供的可選參數，StreamProcessorName是串流處理器資源的名稱，以及SessionId是串流處理工作階段的唯一 ID。根據您的情況更換這些。

偵測自訂標籤

Amazon Rekognition 自訂標籤可以識別出映像中專屬於您業務需求的物體和場景，例如標誌或工程機具零件。如需詳細資訊，請參閱《Amazon Rekognition 自訂標籤開發人員指南》中的什麼是 Amazon Rekognition 自訂標籤？ 。

偵測與分析人臉

Amazon Rekognition 提供您可用來偵測和分析影像和影片中人臉的 API。本節提供用於臉部分析的非儲存操作概觀。這些操作包括檢測面部標誌，分析情緒和比較臉部等功能。

Amazon Rekognition 可識別臉部地標 (例如眼睛位置)、偵測情緒 (例如快樂或悲傷) 和其他屬性 (例如眼鏡存在、臉部遮擋)。偵測到臉部時，系統會分析臉部屬性，並傳回每個屬性的置信度分數。

本節包含影像和視訊操作的範例。

如需有關使用 Rekognition 影像作業的詳細資訊，請參閱。使用映像

如需有關使用 Rekognition 視訊作業的詳細資訊，請參閱。使用儲存的影片分析

請注意，這些操作是非存儲操作。您可以使用儲存空間操作和臉部集合，為影像中偵測到的臉孔儲存臉部中繼資料。您之後可以搜尋存放在映像與影片中的人臉。例如，您可以在影片中搜尋特定人員。如需詳細資訊，請參閱 在集合中搜尋人臉。

如需詳細資訊，請參閱 Amazon Rekognition 常見問題集中的臉孔部分。

> **Note**
>
> Amazon Rekognition Image 和 Amazon Rekognition Video 使用的人臉偵測模型不支援偵測卡通/動畫人物或非人類實體。如果您想要偵測映像或影片中的卡通人物，我們建議您使用 Amazon Rekognition 自訂標籤。如需詳細資訊，請參閱 Amazon Rekognition 自訂標籤開發人員指南》。

主題

人臉偵測和人臉辨比較的概觀

Amazon Rekognition 可讓使用者存取兩個主要機器學習應用程式，以存取包含臉部的影像：臉部偵測和臉部比較。它們增強了面部分析和身份驗證等關鍵功能，使其對於從安全性到個人照片整理的各種應用程序至關重要。

臉部偵測

人臉檢測系統解決了這個問題：「這張照片中有臉嗎？」 人臉檢測的關鍵方面包括：

- 位置和方向：決定影像或視訊影格中面孔的存在、位置、比例和方向。
- 臉部屬性：無論性別，年齡或面部毛髮等屬性如何，都可以檢測臉部。
- 其他資訊：提供臉部遮擋和眼睛凝視方向的詳細資訊。

臉部比較

臉部比較系統專注於這個問題：「一張圖像中的臉是否與另一個圖像中的臉部匹配？」 臉部比較系統功能包括：

- 臉部比對預測：將影像中的臉部與提供的資料庫中的臉孔進行比較，以預測相符項目。
- 臉部屬性處理：處理屬性以比較臉部，無論表情，面部毛髮和年齡如何。

信心分數和遺漏偵測

人臉檢測和臉部比較系統都使用可信度分數。置信度分數表示預測的可能性，例如存在臉孔或臉孔之間相符。分數越高表示可能性越大。例如，90% 的信賴度表示正確偵測或比對的可能性高於 60%。

如果臉部偵測系統無法正確偵測臉部，或對實際臉部提供低置信度預測，這就是遺漏偵測/假陰性。如果系統在高置信度下錯誤地預測了臉部的存在，則這是誤報/誤報。

同樣，臉部比較系統可能無法匹配屬於同一個人的兩張臉孔（錯過檢測/假陰性），或者可能不正確地預測來自不同人的兩張臉孔是同一個人（假警報/假陽性）。

應用程式設計與門檻設定

- 您可以設定閾值，以指定傳回結果所需的最低信賴等級。選擇適當的信賴閾值對於基於系統輸出的應用程序設計和決策至關重要。
- 您選擇的信心等級應反映您的使用案例。使用案例和可信度閾值的一些範例：
- • 照片應用程序：較低的閾值（例如 80%）可能足以識別照片中的家庭成員。
  - 高風險案例：在遺漏偵測或誤報風險較高的使用案例中，例如安全性應用程式，系統應該使用較高的可信度等級。在這種情況下，建議使用更高的閾值（例如 99%）以進行準確的面部比賽。

如需設定和瞭解信賴度閾值的詳細資訊，請參閱在集合中搜尋人臉。

人臉屬性的準則

以下是有關 Amazon Rekognition 如何處理和返回面部屬性的具體細節。

- FaceDetail 對象：對於每個檢測到的臉部，返回一個 FaceDetail 對象。這 FaceDetail 包含有關人臉標記，質量，姿勢等的數據。
- 屬性預測：預測情感，性別，年齡和其他屬性。系統會為每個預測指派信賴等級，並傳回各自的可信度分數的預測。建議敏感使用案例使用 99% 的信賴閾值。對於年齡估算，預測年齡範圍的中點提供了最佳的近似值。

請注意，性別和情緒預測基於外表，不應用於確定實際的性別認同或情感狀態。性別二元論 (男性/女性) 預測是根據特定映像中人臉的實體外觀來判定。它不表示一個人的性別認同，你不應該使用 Rekognition 做出這樣的決心。我們不建議採用性別二元論預測來制定會影響個人權利、隱私或服務存取的決策。同樣，對情感的預測並不表明一個人的實際內部情緒狀態，您不應該使用 Rekognition 做出這樣的決心。一個假裝在照片中擁有幸福的臉的人可能看起來很開心，但可能沒有經歷幸福。

應用與使用案例

以下是這些屬性的一些實際應用和使用案例：

- 應用：微笑，姿勢和清晰度等屬性可用於選擇個人資料照片或匿名估計人口統計數據。
- 常見使用案例：社交媒體應用程序和活動或零售商店的人口統計估計是典型的例子。

如需每個屬性的詳細資訊，請參閱FaceDetail。

偵測映像中的人臉

Amazon Rekognition Image 提供的DetectFaces操作可尋找關鍵臉部特徵 (例如眼睛、鼻子和嘴巴)，以偵測輸入影像中的臉部。Amazon Rekognition Image 可偵測映像中 100 張最大的臉孔。

您可以用映像位元組陣列 (Base64 編碼映像位元組) 的方式提供輸入映像，或指定 Amazon S3 物件。在此程序中，您會將映像 (JPEG 或 PNG) 上傳至您的 S3 儲存貯體，並指定物件索引碼名稱。

偵測映像中的人臉

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 將 (含有一或多個人臉) 的映像上傳至您的 S3 儲存貯體。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例以呼叫 DetectFaces。

   Java

   此範例顯示偵測到之人臉的估計年齡範圍，並列出所有偵測到之人臉屬性的 JSON。變更映像檔案名稱的 photo 值。變更儲存映像之 Amazon S3 儲存貯體的 bucket 值。

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

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

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


public class DetectFaces {


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

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

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


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

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

         for (FaceDetail face: faceDetails) {
            if (request.getAttributes().contains("ALL")) {
               AgeRange ageRange = face.getAgeRange();
```

```
                System.out.println("The detected face is estimated to be between
 "
                    + ageRange.getLow().toString() + " and " +
 ageRange.getHigh().toString()
                    + " years old.");
                System.out.println("Here's the complete set of attributes:");
             } else { // non-default attributes have null values.
                System.out.println("Here's the default set of attributes:");
             }

             ObjectMapper objectMapper = new ObjectMapper();

 System.out.println(objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(fac
          }

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

    }

}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import java.util.List;

//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.AgeRange;
```

```
//snippet-end:[rekognition.java2.detect_labels.import]

public class DetectFaces {

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage: " +
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,ImageBucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

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

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

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

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    public static void getLabelsfromImage(RekognitionClient rekClient, String
 bucket, String image) {

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

            Image myImage = Image.builder()
                .s3Object(s3Object)
```

```
                .build();

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

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

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

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
 // snippet-end:[rekognition.java2.detect_labels.main]
}
```

AWS CLI

此範例會顯示detect-faces AWS CLI 作業的 JSON 輸出。將 file 取代為映像檔案的名稱。將 bucket 取代為 Amazon S3 儲存貯體的名稱，其中包含映像檔案。

```
aws rekognition detect-faces --image "{"S3Object":{"Bucket":"bucket-
name","Name":"image-name"}}"\
                            --attributes "ALL" --profile profile-name --region
 region-name
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
aws rekognition detect-faces --image "{\"S3Object\":{\"Bucket\":\"bucket-name\",
\"Name\":\"image-name\"}}" --attributes "ALL"
--profile profile-name --region region-name
```

Python

此範例顯示偵測到之人臉的估計年齡範圍，並列出所有偵測到之人臉屬性的 JSON。變更映像檔案名稱的 photo 值。變更儲存映像之 Amazon S3 儲存貯體的 bucket 值。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
import boto3
import json

def detect_faces(photo, bucket, region):

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

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

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

        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))

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

```
        print("Face Occluded: " + str(faceDetail['FaceOccluded']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])

def main():
    photo='photo'
    bucket='bucket'
    region='region'
    face_count=detect_faces(photo, bucket, region)
    print("Faces detected: " + str(face_count))

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

.NET

此範例顯示偵測到之人臉的估計年齡範圍，並列出所有偵測到之人臉屬性的 JSON。變更映像檔案名稱的 photo 值。變更儲存映像之 Amazon S3 儲存貯體的 bucket 值。

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectFacesRequest detectFacesRequest = new DetectFacesRequest()
        {
            Image = new Image()
            {
```

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

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

Ruby

此範例顯示偵測到之人臉的估計年齡範圍，並列出各種人臉屬性。變更映像檔案名稱的 photo 值。變更儲存映像之 Amazon S3 儲存貯體的 bucket 值。

```
# Add to your Gemfile
# gem 'aws-sdk-rekognition'
require 'aws-sdk-rekognition'
credentials = Aws::Credentials.new(
   ENV['AWS_ACCESS_KEY_ID'],
   ENV['AWS_SECRET_ACCESS_KEY']
)
bucket = 'bucket' # the bucketname without s3://
photo  = 'input.jpg'# the name of file
client   = Aws::Rekognition::Client.new credentials: credentials
attrs = {
  image: {
    s3_object: {
      bucket: bucket,
      name: photo
    },
  },
  attributes: ['ALL']
}
response = client.detect_faces attrs
puts "Detected faces for: #{photo}"
response.face_details.each do |face_detail|
  low  = face_detail.age_range.low
  high = face_detail.age_range.high
  puts "The detected face is between: #{low} and #{high} years old"
  puts "All other attributes:"
  puts "  bounding_box.width:     #{face_detail.bounding_box.width}"
  puts "  bounding_box.height:    #{face_detail.bounding_box.height}"
  puts "  bounding_box.left:      #{face_detail.bounding_box.left}"
  puts "  bounding_box.top:       #{face_detail.bounding_box.top}"
  puts "  age.range.low:          #{face_detail.age_range.low}"
  puts "  age.range.high:         #{face_detail.age_range.high}"
  puts "  smile.value:            #{face_detail.smile.value}"
  puts "  smile.confidence:       #{face_detail.smile.confidence}"
  puts "  eyeglasses.value:       #{face_detail.eyeglasses.value}"
  puts "  eyeglasses.confidence:  #{face_detail.eyeglasses.confidence}"
```

```
    puts "  sunglasses.value:       #{face_detail.sunglasses.value}"
    puts "  sunglasses.confidence:  #{face_detail.sunglasses.confidence}"
    puts "  gender.value:           #{face_detail.gender.value}"
    puts "  gender.confidence:      #{face_detail.gender.confidence}"
    puts "  beard.value:            #{face_detail.beard.value}"
    puts "  beard.confidence:       #{face_detail.beard.confidence}"
    puts "  mustache.value:         #{face_detail.mustache.value}"
    puts "  mustache.confidence:    #{face_detail.mustache.confidence}"
    puts "  eyes_open.value:        #{face_detail.eyes_open.value}"
    puts "  eyes_open.confidence:   #{face_detail.eyes_open.confidence}"
    puts "  mout_open.value:        #{face_detail.mouth_open.value}"
    puts "  mout_open.confidence:   #{face_detail.mouth_open.confidence}"
    puts "  emotions[0].type:       #{face_detail.emotions[0].type}"
    puts "  emotions[0].confidence: #{face_detail.emotions[0].confidence}"
    puts "  landmarks[0].type:      #{face_detail.landmarks[0].type}"
    puts "  landmarks[0].x:         #{face_detail.landmarks[0].x}"
    puts "  landmarks[0].y:         #{face_detail.landmarks[0].y}"
    puts "  pose.roll:              #{face_detail.pose.roll}"
    puts "  pose.yaw:               #{face_detail.pose.yaw}"
    puts "  pose.pitch:             #{face_detail.pose.pitch}"
    puts "  quality.brightness:     #{face_detail.quality.brightness}"
    puts "  quality.sharpness:      #{face_detail.quality.sharpness}"
    puts "  confidence:             #{face_detail.confidence}"
    puts "------------"
    puts ""
  end
```

Node.js

此範例顯示偵測到之人臉的估計年齡範圍，並列出各種人臉屬性。變更映像檔案名稱的 photo 值。變更儲存映像之 Amazon S3 儲存貯體的 bucket 值。

將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

如果您正在使用 TypeScript 定義，您可能需要使用`import AWS from 'aws-sdk'`而不是`const AWS = require('aws-sdk')`，以便使用 Node.js 運行該程序。您可以查閱適用於 Javascript 的AWS SDK，以獲取更多詳細資訊。視您設定組態的方式而定，您可能還需要使用 `AWS.config.update({region:`*region*`});` 來指定您所在的區域。

```
// Load the SDK
```

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

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

const client = new AWS.Rekognition();
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
  Attributes: ['ALL']
}

client.detectFaces(params, function(err, response) {
    if (err) {
      console.log(err, err.stack); // an error occurred
    } else {
      console.log(`Detected faces for: ${photo}`)
      response.FaceDetails.forEach(data => {
        let low  = data.AgeRange.Low
        let high = data.AgeRange.High
        console.log(`The detected face is between: ${low} and ${high} years
 old`)
        console.log("All other attributes:")
        console.log(`  BoundingBox.Width:      ${data.BoundingBox.Width}`)
        console.log(`  BoundingBox.Height:     ${data.BoundingBox.Height}`)
        console.log(`  BoundingBox.Left:       ${data.BoundingBox.Left}`)
        console.log(`  BoundingBox.Top:        ${data.BoundingBox.Top}`)
        console.log(`  Age.Range.Low:          ${data.AgeRange.Low}`)
        console.log(`  Age.Range.High:         ${data.AgeRange.High}`)
        console.log(`  Smile.Value:            ${data.Smile.Value}`)
        console.log(`  Smile.Confidence:       ${data.Smile.Confidence}`)
        console.log(`  Eyeglasses.Value:       ${data.Eyeglasses.Value}`)
        console.log(`  Eyeglasses.Confidence:  ${data.Eyeglasses.Confidence}`)
        console.log(`  Sunglasses.Value:       ${data.Sunglasses.Value}`)
        console.log(`  Sunglasses.Confidence:  ${data.Sunglasses.Confidence}`)
        console.log(`  Gender.Value:           ${data.Gender.Value}`)
        console.log(`  Gender.Confidence:      ${data.Gender.Confidence}`)
```

```
        console.log(`  Beard.Value:             ${data.Beard.Value}`)
        console.log(`  Beard.Confidence:        ${data.Beard.Confidence}`)
        console.log(`  Mustache.Value:          ${data.Mustache.Value}`)
        console.log(`  Mustache.Confidence:     ${data.Mustache.Confidence}`)
        console.log(`  EyesOpen.Value:          ${data.EyesOpen.Value}`)
        console.log(`  EyesOpen.Confidence:     ${data.EyesOpen.Confidence}`)
        console.log(`  MouthOpen.Value:         ${data.MouthOpen.Value}`)
        console.log(`  MouthOpen.Confidence:    ${data.MouthOpen.Confidence}`)
        console.log(`  Emotions[0].Type:        ${data.Emotions[0].Type}`)
        console.log(`  Emotions[0].Confidence: ${data.Emotions[0].Confidence}`)
        console.log(`  Landmarks[0].Type:       ${data.Landmarks[0].Type}`)
        console.log(`  Landmarks[0].X:          ${data.Landmarks[0].X}`)
        console.log(`  Landmarks[0].Y:          ${data.Landmarks[0].Y}`)
        console.log(`  Pose.Roll:               ${data.Pose.Roll}`)
        console.log(`  Pose.Yaw:                ${data.Pose.Yaw}`)
        console.log(`  Pose.Pitch:              ${data.Pose.Pitch}`)
        console.log(`  Quality.Brightness:      ${data.Quality.Brightness}`)
        console.log(`  Quality.Sharpness:       ${data.Quality.Sharpness}`)
        console.log(`  Confidence:              ${data.Confidence}`)
        console.log("------------")
        console.log("")
      }) // for response.faceDetails
    } // if
  });
```

DetectFaces 操作請求

DetectFaces 的輸入是映像。在此範例中，映像是從 Amazon S3 儲存貯體載入的。Attributes 參數指定所有臉部屬性應該要傳回。如需詳細資訊，請參閱 使用映像。

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

DetectFaces 作業回應

DetectFaces 針對每個偵測到的臉孔傳回以下資訊：

- 週框方塊：人臉周圍的週框方塊座標。
- 可信度：週框方塊包含人臉的可信度。
- 人臉特徵點：人臉特徵點的陣列。對於每個特徵點 (例如左眼、右眼與嘴巴)，回應均會提供 x 和 y 座標。
- 人臉屬性：一組人臉屬性，例如人臉是否被遮擋，以 FaceDetail 的物件形式傳回。該套裝包括：鬍子 AgeRange，情感 EyeDirection，眼鏡，， EyesOpen，性別 FaceOccluded，鬍子 MouthOpen，微笑和太陽鏡。對於每個這類屬性，該回應均會提供一個值。該值的類型可能不同，例如布林值類型 (此人是否戴著太陽眼鏡) 或字串類型 (此人是男性或女性) 或 (眼睛注視方向的俯仰/偏移) 的角度值類型。此外，對於大部分屬性，該回應也會提供偵測到之屬性值的可信度。請注意，雖 FaceOccluded 然使用時支持和 EyeDirection 屬性DetectFaces，但使用StartFaceDetection和分析視頻時不支持它們GetFaceDetection。
- 畫質：描述人臉的亮度與銳利度。如需盡可能確保最佳臉部偵測的資訊，請參閱 人臉比較輸入映像的建議。
- 姿態：描述映像內人臉的旋轉。

要求可以描繪您想要傳回的人臉屬性陣列。人臉屬性的 DEFAULT 子集：始終傳回 BoundingBox Confidence、Pose、Quality 和 Landmarks。您可以請求傳回特定的人臉屬性 (除了預設清單)，透過使用 ["DEFAULT", "FACE_OCCLUDED", "EYE_DIRECTION"] 或僅使用一個屬性，例如 ["FACE_OCCLUDED"]。您可以使用 ["ALL"] 要求所有人臉屬性。要求更多屬性可能會增加回應時間。

以下是 DetectFaces API 呼叫的回應範例。

```
{
  "FaceDetails": [
    {
      "BoundingBox": {
        "Width": 0.7919622659683228,
        "Height": 0.7510867118835449,
        "Left": 0.08881539851427078,
        "Top": 0.151064932346344
      },
      "AgeRange": {
```

```
        "Low": 18,
        "High": 26
      },
      "Smile": {
        "Value": false,
        "Confidence": 89.77348327636719
      },
      "Eyeglasses": {
        "Value": true,
        "Confidence": 99.99996948242188
      },
      "Sunglasses": {
        "Value": true,
        "Confidence": 93.65237426757812
      },
      "Gender": {
        "Value": "Female",
        "Confidence": 99.85968780517578
      },
      "Beard": {
        "Value": false,
        "Confidence": 77.52591705322266
      },
      "Mustache": {
        "Value": false,
        "Confidence": 94.48904418945312
      },
      "EyesOpen": {
        "Value": true,
        "Confidence": 98.57169342041016
      },
      "MouthOpen": {
        "Value": false,
        "Confidence": 74.33953094482422
      },
      "Emotions": [
        {
          "Type": "SAD",
          "Confidence": 65.56403350830078
        },
        {
          "Type": "CONFUSED",
          "Confidence": 31.277774810791016
        },
```

```
        {
          "Type": "DISGUSTED",
          "Confidence": 15.553778648376465
        },
        {
          "Type": "ANGRY",
          "Confidence": 8.012762069702148
        },
        {
          "Type": "SURPRISED",
          "Confidence": 7.621500015258789
        },
        {
          "Type": "FEAR",
          "Confidence": 7.243380546569824
        },
        {
          "Type": "CALM",
          "Confidence": 5.8196024894714355
        },
        {
          "Type": "HAPPY",
          "Confidence": 2.2830512523651123
        }
      ],
      "Landmarks": [
        {
          "Type": "eyeLeft",
          "X": 0.30225440859794617,
          "Y": 0.41018882393836975
        },
        {
          "Type": "eyeRight",
          "X": 0.6439348459243774,
          "Y": 0.40341562032699585
        },
        {
          "Type": "mouthLeft",
          "X": 0.343580037355423,
          "Y": 0.6951127648353577
        },
        {
          "Type": "mouthRight",
          "X": 0.6306480765342712,
```

```
          "Y": 0.6898072361946106
        },
        {
          "Type": "nose",
          "X": 0.47164231538772583,
          "Y": 0.5763645172119141
        },
        {
          "Type": "leftEyeBrowLeft",
          "X": 0.1732882857322693,
          "Y": 0.34452149271965027
        },
        {
          "Type": "leftEyeBrowRight",
          "X": 0.3655243515968323,
          "Y": 0.33231860399246216
        },
        {
          "Type": "leftEyeBrowUp",
          "X": 0.2671719491481781,
          "Y": 0.31669262051582336
        },
        {
          "Type": "rightEyeBrowLeft",
          "X": 0.5613729953765869,
          "Y": 0.32813435792922974
        },
        {
          "Type": "rightEyeBrowRight",
          "X": 0.7665090560913086,
          "Y": 0.3318614959716797
        },
        {
          "Type": "rightEyeBrowUp",
          "X": 0.6612788438796997,
          "Y": 0.3082450032234192
        },
        {
          "Type": "leftEyeLeft",
          "X": 0.2416982799768448,
          "Y": 0.4085965156555176
        },
        {
          "Type": "leftEyeRight",
```

```
          "X": 0.36943578720092773,
          "Y": 0.41230902075767517
        },
        {
          "Type": "leftEyeUp",
          "X": 0.29974061250686646,
          "Y": 0.3971870541572571
        },
        {
          "Type": "leftEyeDown",
          "X": 0.30360740423202515,
          "Y": 0.42347756028175354
        },
        {
          "Type": "rightEyeLeft",
          "X": 0.5755768418312073,
          "Y": 0.4081145226955414
        },
        {
          "Type": "rightEyeRight",
          "X": 0.7050536870956421,
          "Y": 0.39924031496047974
        },
        {
          "Type": "rightEyeUp",
          "X": 0.642906129360199,
          "Y": 0.39026668667793274
        },
        {
          "Type": "rightEyeDown",
          "X": 0.6423097848892212,
          "Y": 0.41669243574142456
        },
        {
          "Type": "noseLeft",
          "X": 0.4122826159000397,
          "Y": 0.5987403392791748
        },
        {
          "Type": "noseRight",
          "X": 0.5394935011863708,
          "Y": 0.5960900187492371
        },
        {
```

```
          "Type": "mouthUp",
          "X": 0.478581964969635,
          "Y": 0.6660456657409668
        },
        {
          "Type": "mouthDown",
          "X": 0.483366996049881,
          "Y": 0.7497162818908691
        },
        {
          "Type": "leftPupil",
          "X": 0.30225440859794617,
          "Y": 0.41018882393836975
        },
        {
          "Type": "rightPupil",
          "X": 0.6439348459243774,
          "Y": 0.40341562032699585
        },
        {
          "Type": "upperJawlineLeft",
          "X": 0.11031254380941391,
          "Y": 0.3980775475502014
        },
        {
          "Type": "midJawlineLeft",
          "X": 0.19301874935626984,
          "Y": 0.7034031748771667
        },
        {
          "Type": "chinBottom",
          "X": 0.4939905107021332,
          "Y": 0.8877836465835571
        },
        {
          "Type": "midJawlineRight",
          "X": 0.7990140914916992,
          "Y": 0.6899225115776062
        },
        {
          "Type": "upperJawlineRight",
          "X": 0.8548634648323059,
          "Y": 0.38160091638565063
        }
```

```
      ],
      "Pose": {
        "Roll": -5.83309268951416,
        "Yaw": -2.4244730472564697,
        "Pitch": 2.6216139793395996
      },
      "Quality": {
        "Brightness": 96.16363525390625,
        "Sharpness": 95.51618957519531
      },
      "Confidence": 99.99872589111328,
      "FaceOccluded": {
        "Value": true,
        "Confidence": 99.99726104736328
      },
      "EyeDirection": {
        "Yaw": 16.299732,
        "Pitch": -6.407457,
        "Confidence": 99.968704
      }
    }
  ],
  "ResponseMetadata": {
    "RequestId": "8bf02607-70b7-4f20-be55-473fe1bba9a2",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "8bf02607-70b7-4f20-be55-473fe1bba9a2",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "3409",
      "date": "Wed, 26 Apr 2023 20:18:50 GMT"
    },
    "RetryAttempts": 0
  }
}
```


注意下列事項：

- Pose 資料描述偵測到之臉部的旋轉。您可以使用 BoundingBox 與 Pose 資料的組合，在您的應用程式所顯示的臉部周圍繪製週框方塊。
- Quality 描述臉部的亮度與銳利度。您可能會發現這有助於在映像之間比較人臉，並找出最佳人臉。

- 上述回應顯示服務可偵測到的所有臉部 landmarks、所有臉部屬性與情緒。若要在回應中取得所有專案，您必須將 attributes 參數的值指定為 ALL。根據預設，DetectFaces API 只會傳回下列五個臉部屬性：BoundingBox、Confidence、Pose、Quality 和 landmarks。傳回的預設特徵點如下：eyeLeft、eyeRight、nose、mouthLeft 和 mouthRight。

比較映像中的人臉

使用 Rekognition，您可以使用操作比較兩個影像之間的臉孔。CompareFaces此功能對於身份驗證或照片匹配等應用程序非常有用。

CompareFaces 將來源影像中的臉孔與目標影像中的每個臉孔進行比較。圖像 CompareFaces 作為以下任一方式傳遞給：

- 圖像的 Base64 編碼表示。
- Amazon S3 對象。

人臉檢測與臉部比較

臉部比較與臉部偵測不同。臉部偵測 (使用 DetectFaces) 僅識別影像或視訊中臉孔的存在與位置。相比之下，臉部比較包括將來源影像中偵測到的臉孔與目標影像中的臉孔進行比較，以尋找相符項目。

相似性閾值

使用similarityThreshold參數可定義要包含在回應中的相符項目的最小信賴等級。依預設，回應中只會傳回相似度分數大於或等於 80% 的臉孔。

> **Note**
>
> CompareFaces使用機器學習算法，這是概率的。假陰性指的是不正確的預測，指出目標映像中的人臉與來源映像中的人臉相比具有較低的相似度置信度分數。為了減少偽陰性的可能性，建議您將目標映像與多個來源映像進行比較。如果您打算使用 CompareFaces 做出會影響個人權利、隱私權或服務存取權的決定，我們建議您在採取行動之前，將結果傳遞給人員進行審查並進一步驗證。

下列程式碼範例示範如何針對各種 AWS SDK 使用這些 CompareFaces 作業。在此 AWS CLI 範例中，您將兩個 JPEG 影像上傳到 Amazon S3 儲存貯體，並指定物件金鑰名稱。在其他範例中，您會從本機檔案系統載入兩個檔案，並以映像位元組陣列的方式加以輸入。

比較臉部

1. 如果您尚未執行：

   a. 建立或更新具有AmazonRekognitionFullAccess和 AmazonS3ReadOnlyAccess (僅限 AWS CLI 範例) 權限的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列程式碼範例來呼叫 CompareFaces 操作。

   Java

   此範例顯示有關來源和目標映像 (從本機檔案系統載入) 中相符臉孔的資訊。

   將 sourceImage 與 targetImage 的值取代為來源和目標映像的路徑和檔案名稱。

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

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

public class CompareFaces {
```

```
  public static void main(String[] args) throws Exception{
      Float similarityThreshold = 70F;
      String sourceImage = "source.jpg";
      String targetImage = "target.jpg";
      ByteBuffer sourceImageBytes=null;
      ByteBuffer targetImageBytes=null;

      AmazonRekognition rekognitionClient =
AmazonRekognitionClientBuilder.defaultClient();

      //Load source and target images and create input parameters
      try (InputStream inputStream = new FileInputStream(new
File(sourceImage))) {
         sourceImageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
      }
      catch(Exception e)
      {
          System.out.println("Failed to load source image " + sourceImage);
          System.exit(1);
      }
      try (InputStream inputStream = new FileInputStream(new
File(targetImage))) {
          targetImageBytes =
ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
      }
      catch(Exception e)
      {
          System.out.println("Failed to load target images: " + targetImage);
          System.exit(1);
      }

      Image source=new Image()
           .withBytes(sourceImageBytes);
      Image target=new Image()
           .withBytes(targetImageBytes);

      CompareFacesRequest request = new CompareFacesRequest()
              .withSourceImage(source)
              .withTargetImage(target)
              .withSimilarityThreshold(similarityThreshold);

      // Call operation
```

```
      CompareFacesResult
 compareFacesResult=rekognitionClient.compareFaces(request);


      // Display results
      List <CompareFacesMatch> faceDetails =
 compareFacesResult.getFaceMatches();
      for (CompareFacesMatch match: faceDetails){
        ComparedFace face= match.getFace();
        BoundingBox position = face.getBoundingBox();
        System.out.println("Face at " + position.getLeft().toString()
              + " " + position.getTop()
              + " matches with " + match.getSimilarity().toString()
              + "% confidence.");

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

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

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import java.util.List;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import software.amazon.awssdk.services.rekognition.model.CompareFacesMatch;
import software.amazon.awssdk.services.rekognition.model.CompareFacesRequest;
import software.amazon.awssdk.services.rekognition.model.CompareFacesResponse;
import software.amazon.awssdk.services.rekognition.model.ComparedFace;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
```

```
// snippet-end:[rekognition.java2.detect_faces.import]

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

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <pathSource> <pathTarget>\n\n" +
            "Where:\n" +
            "   pathSource - The path to the source image (for example, C:\\AWS\
\pic1.png). \n " +
            "   pathTarget - The path to the target image (for example, C:\\AWS\
\pic2.png). \n\n";

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

        Float similarityThreshold = 70F;
        String sourceImage = args[0];
        String targetImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        compareTwoFaces(rekClient, similarityThreshold, sourceImage,
 targetImage);
        rekClient.close();
   }
```

```
   // snippet-start:[rekognition.java2.compare_faces.main]
   public static void compareTwoFaces(RekognitionClient rekClient, Float
similarityThreshold, String sourceImage, String targetImage) {
       try {
           InputStream sourceStream = new FileInputStream(sourceImage);
           InputStream tarStream = new FileInputStream(targetImage);
           SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
           SdkBytes targetBytes = SdkBytes.fromInputStream(tarStream);

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

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

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

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

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

```
            System.out.println("target image rotation: " +
 compareFacesResult.targetImageOrientationCorrection());

        } catch(RekognitionException | FileNotFoundException e) {
            System.out.println("Failed to load source image " + sourceImage);
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.compare_faces.main]
}
```

AWS CLI

此範例會顯示compare-faces AWS CLI 作業的 JSON 輸出。

將 bucket-name 取代為 Amazon S3 儲存貯體的名稱，其中包含來源和目標映像。將 source.jpg 和 target.jpg 取代為來源和目標映像的檔案名稱。

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

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

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

Python

此範例顯示有關來源和目標映像 (從本機檔案系統載入) 中相符臉孔的資訊，表面載入在從本機檔案系統。

將 source_file 與 target_file 的值取代為來源和目標映像的路徑和檔案名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3

def compare_faces(sourceFile, targetFile):

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

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

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

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

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

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

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

.NET

此範例顯示有關來源和目標映像 (從本機檔案系統載入) 中相符臉孔的資訊。

將 sourceImage 與 targetImage 的值取代為來源和目標映像的路徑和檔案名稱。

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

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

        Amazon.Rekognition.Model.Image imageTarget = new
 Amazon.Rekognition.Model.Image();
        try
```

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

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

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

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

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

    }
}
```

Ruby

此範例顯示有關來源和目標映像 (從本機檔案系統載入) 中相符臉孔的資訊，表面載入在從本機檔案系統。

將 photo_source 與 photo_target 的值取代為來源和目標映像的路徑和檔案名稱。

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

Node.js

此範例顯示有關來源和目標映像 (從本機檔案系統載入) 中相符臉孔的資訊，表面載入在從本機檔案系統。

將 photo_source 與 photo_target 的值取代為來源和目標映像的路徑和檔案名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
// Load the SDK
var AWS = require('aws-sdk');
const bucket = 'bucket-name' // the bucket name without s3://
const photo_source  = 'photo-source-name' // path and the name of file
const photo_target = 'photo-target-name'

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

const client = new AWS.Rekognition();
   const params = {
     SourceImage: {
       S3Object: {
         Bucket: bucket,
         Name: photo_source
       },
     },
     TargetImage: {
       S3Object: {
         Bucket: bucket,
         Name: photo_target
       },
     },
     SimilarityThreshold: 70
   }
   client.compareFaces(params, function(err, response) {
     if (err) {
       console.log(err, err.stack); // an error occurred
     } else {
       response.FaceMatches.forEach(data => {
         let position   = data.Face.BoundingBox
         let similarity = data.Similarity
```

```
        console.log(`The face at: ${position.Left}, ${position.Top} matches
with ${similarity} % confidence`)
      }) // for response.faceDetails
    } // if
  });
```

CompareFaces 操作請求

CompareFaces 的輸入是映像。在此範例中，來源和目標映像從本機檔案系統載入。SimilarityThreshold 輸入參數指定比較臉孔必須符合的最低可信度應包含在回應中。如需詳細資訊，請參閱 使用映像。

```
{
    "SourceImage": {
        "Bytes": "/9j/4AAQSk2Q==..."
    },
    "TargetImage": {
        "Bytes": "/9j/4O1Q==..."
    },
    "SimilarityThreshold": 70
}
```

CompareFaces 作業回應

回應包括：

- 臉孔相符的陣列：符合臉孔的清單，其中包含相似度分數和每個相符臉孔的中繼資料。如果多個面相符，faceMatches

  陣列包括所有的臉部相符項目。
- 臉孔比對詳細資料：每個相符的臉孔也會提供邊界方框、置信度值、地標位置和相似度分數。
- 不相符的臉孔清單：回應也包含目標影像中與來源影像臉孔不相符的臉孔。包括每個無與倫比的面的邊界框。
- 來源臉孔資訊：包括來源影像中用於比較的臉孔的相關資訊，包括邊界方框和置信度值。

此範例顯示在目標影像中找到了一個相符的面。對於該相符的人臉，該回應會提供週框方塊與可信度值 (Amazon Rekognition 對週框方塊包含人臉的可信度)。99.99 的相似度分數表示臉孔有多相似。此範例也會顯示 Amazon Rekognition 在目標影像中找到的一張臉孔，與來源影像中分析的臉孔不符。

```
{
    "FaceMatches": [{
        "Face": {
            "BoundingBox": {
                "Width": 0.5521978139877319,
                "Top": 0.1203877404332161,
                "Left": 0.23626373708248138,
                "Height": 0.3126954436302185
            },
            "Confidence": 99.98751068115234,
            "Pose": {
                "Yaw": -82.36799621582031,
                "Roll": -62.13221740722656,
                "Pitch": 0.8652129173278809
            },
            "Quality": {
                "Sharpness": 99.99880981445312,
                "Brightness": 54.49755096435547
            },
            "Landmarks": [{
                    "Y": 0.2996366024017334,
                    "X": 0.41685718297958374,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.2658946216106415,
                    "X": 0.4414493441581726,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3465650677680969,
                    "X": 0.48636093735694885,
                    "Type": "nose"
                },
                {
                    "Y": 0.30935320258140564,
                    "X": 0.6251809000968933,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.26942989230155945,
                    "X": 0.6454493403434753,
                    "Type": "mouthRight"
```

```
                }
            ]
        },
        "Similarity": 100.0
    }],
    "SourceImageOrientationCorrection": "ROTATE_90",
    "TargetImageOrientationCorrection": "ROTATE_90",
    "UnmatchedFaces": [{
        "BoundingBox": {
            "Width": 0.4890109896659851,
            "Top": 0.6566604375839233,
            "Left": 0.10989011079072952,
            "Height": 0.278298944234848
        },
        "Confidence": 99.99992370605469,
        "Pose": {
            "Yaw": 51.51519012451172,
            "Roll": -110.32493591308594,
            "Pitch": -2.322134017944336
        },
        "Quality": {
            "Sharpness": 99.99671173095703,
            "Brightness": 57.23163986206055
        },
        "Landmarks": [{
                "Y": 0.8288310766220093,
                "X": 0.3133862614631653,
                "Type": "eyeLeft"
            },
            {
                "Y": 0.7632885575294495,
                "X": 0.28091415762901306,
                "Type": "eyeRight"
            },
            {
                "Y": 0.7417283654212952,
                "X": 0.3631140887737274,
                "Type": "nose"
            },
            {
                "Y": 0.8081989884376526,
                "X": 0.4856561422348025,
                "Type": "mouthLeft"
            },
```

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


偵測已存放影片中的人臉

Amazon Rekognition Video 可以偵測存放在 Amazon S3 儲存貯體之影片中的人臉，並提供下列資訊：

- 在影片中偵測到臉部的次數。
- 偵測到臉部時臉部在影片影格中的位置。
- 臉部特徵點，例如左眼的位置。
- 其他屬性，如 the section called “人臉屬性的準則” 頁面上所述。

Amazon Rekognition Video 已儲存影片中的人臉偵測是一項非同步操作。要開始檢測視頻中的臉部，請致電StartFaceDetection。Amazon Rekognition Video 會將影片分析的完成狀态发布至 Amazon Simple Notification Service (Amazon SNS) 主題。如果視頻分析成功，則可以調用GetFaceDetection以獲取視頻分析的結果。如需開始影片分析並取得結果的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

此程序會展開 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中的程式碼，該操作使用 Amazon Simple Queue Service (Amazon SQS) 佇列來取得影片分析要求的完成狀態。

偵測存放在 Amazon S3 儲存貯體 (SDK) 之影片中的人臉

1. 執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。

2. 將下列程式碼新增至您在步驟 1 中建立的類別 VideoDetect。

AWS CLI

- 在以下程式碼範例中，將 bucket-name 與 video-name 變更為您在步驟 2 中指定的 Amazon S3 儲存貯體與文檔名稱。
- 將 region-name 變更為您正在使用的 AWS 區域。使用您開發人員設定檔的名稱取代 profile_name 的值。
- 將 TopicARN 變更為您在 設定 Amazon Rekognition Video 步驟 3 建立的 Amazon SNS 主題 ARN。
- 將 RoleARN 變更為您在步驟 7 建立的 設定 Amazon Rekognition Video IAM 服務角色的 ARN。

```
aws rekognition start-face-detection --video "{"S3Object":{"Bucket":"Bucket-
Name","Name":"Video-Name"}}" --notification-channel \
"{"SNSTopicArn":"Topic-ARN","RoleArn":"Role-ARN"}" --region region-name --
profile profile-name
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
aws rekognition start-face-detection --video "{\"S3Object\":{\"Bucket\":
\"Bucket-Name\",\"Name\":\"Video-Name\"}}" --notification-channel \
"{\"SNSTopicArn\":\"Topic-ARN\",\"RoleArn\":\"Role-ARN\"}" --region region-name
 --profile profile-name
```

執行 StartFaceDetection 操作，並取得工作 ID 編號之後，請執行下列 GetFaceDetection 作業並提供工作 ID 號碼：

```
aws rekognition get-face-detection --job-id job-id-number  --profile profile-
name
```

Java

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


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

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

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



    StartFaceDetectionResult startLabelDetectionResult =
 rek.startFaceDetection(req);
    startJobId=startLabelDetectionResult.getJobId();

}

private static void GetFaceDetectionResults() throws Exception{

    int maxResults=10;
    String paginationToken=null;
    GetFaceDetectionResult faceDetectionResult=null;

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

        faceDetectionResult = rek.getFaceDetection(new
 GetFaceDetectionRequest()
             .withJobId(startJobId)
```

```
             .withNextToken(paginationToken)
             .withMaxResults(maxResults));

        VideoMetadata videoMetaData=faceDetectionResult.getVideoMetadata();

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


        //Show faces, confidence and detection times
        List<FaceDetection> faces= faceDetectionResult.getFaces();

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


}
```

在函數 `main` 中，將下行:

```
        StartLabelDetection(bucket, video);

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

取代為：

```
        StartFaceDetection(bucket, video);

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

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.recognize_video_faces.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_faces.import]


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

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

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

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

```
    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

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

    StartFaceDetection(rekClient, channel, bucket, video);
    GetFaceResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_faces.main]
public static void StartFaceDetection(RekognitionClient rekClient,
                                       NotificationChannel channel,
                                       String bucket,
                                       String video) {

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

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

        StartFaceDetectionRequest  faceDetectionRequest =
StartFaceDetectionRequest.builder()
            .jobTag("Faces")
            .faceAttributes(FaceAttributes.ALL)
            .notificationChannel(channel)
```

```
            .video(vidOb)
            .build();

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

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

public static void GetFaceResults(RekognitionClient rekClient) {

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

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

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

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

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

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

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

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null
            VideoMetadata videoMetaData=faceDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

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

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

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

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_faces.main]
}
```

Python

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

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

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

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

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

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

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

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

在函數 main 中，將下行:

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

取代為：

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

> **Note**
>
> 如果您已執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 以外的影片範例，要取代的函數名稱會不同。

3. 執行程式碼。影片中偵測到之臉部的資訊便會顯示。

GetFaceDetection 作業回應

GetFaceDetection 會傳回陣列 (Faces)，其中包含影片中偵測到之臉孔的相關資訊。每次在視訊中偵測到臉孔時，都會存在陣列元素。FaceDetection傳回的陣列元素會依從影片開頭起的時間 (以毫秒為單位) 順序排序。

下列範例是 GetFaceDetection 的部分 JSON 回應。在回應中，請注意下列事項：

- 週框方塊：人臉周圍的週框方塊座標。
- 可信度：週框方塊包含人臉的可信度。
- 人臉特徵點：人臉特徵點的陣列。對於每個特徵點 (例如左眼、右眼與嘴巴)，回應均會提供 x 和 y 座標。
- 臉部屬性 — 一組面部屬性，其中包括：鬍子 AgeRange，情感，眼鏡 EyesOpen，性別，鬍子 MouthOpen，微笑和太陽鏡。該值的類型可能不同，例如布林值類型 (此人是否戴著太陽眼鏡) 或字串 (此人是男性或女性)。此外，對於大部分屬性，該回應也會提供偵測到之屬性值的可信度。請注意，雖 FaceOccluded 然使用時支持和 EyeDirection屬性DetectFaces，但使用StartFaceDetection和分析視頻時不支持它們GetFaceDetection。
- 時間戳記：在影片中偵測到臉孔的時間。
- 分頁資訊：該範例顯示一頁人臉偵測資訊。您可以在 GetFaceDetection 的 MaxResults 輸入參數中指定要傳回的人物元素數目。如果結果數目超過 MaxResults，GetFaceDetection 會傳回用來取得下一頁結果的字符 (NextToken)。如需詳細資訊，請參閱 取得 Amazon Rekognition Video 分析結果。

- 影片資訊：回應包含 GetFaceDetection 所傳回之每頁資訊中影片格式 (VideoMetadata) 的相關資訊。
- 畫質：描述人臉的亮度與銳利度。
- 姿態：描述映像內人臉的旋轉角度。


```
{
    "Faces": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.23000000417232513,
                    "Left": 0.42500001192092896,
                    "Top": 0.16333332657814026,
                    "Width": 0.12937499582767487
                },
                "Confidence": 99.97504425048828,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.46415066719055176,
                        "Y": 0.2572723925113678
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.5068183541297913,
                        "Y": 0.23705792427062988
                    },
                    {
                        "Type": "nose",
                        "X": 0.49765899777412415,
                        "Y": 0.28383663296699524
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.487221896648407,
                        "Y": 0.3452930748462677
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.5142884850502014,
                        "Y": 0.33167609572410583
```

```
                    }
                ],
                "Pose": {
                    "Pitch": 15.966927528381348,
                    "Roll": -15.547388076782227,
                    "Yaw": 11.34195613861084
                },
                "Quality": {
                    "Brightness": 44.80223083496094,
                    "Sharpness": 99.95819854736328
                }
            },
            "Timestamp": 0
        },
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.20000000298023224,
                    "Left": 0.029999999329447746,
                    "Top": 0.2199999988079071,
                    "Width": 0.11249999701976776
                },
                "Confidence": 99.85971069335938,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.06842322647571564,
                        "Y": 0.3010137975215912
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.10543643683195114,
                        "Y": 0.29697132110595703
                    },
                    {
                        "Type": "nose",
                        "X": 0.09569807350635529,
                        "Y": 0.33701086044311523
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.0732642263174057,
                        "Y": 0.3757539987564087
                    },
```

```
                {
                    "Type": "mouthRight",
                    "X": 0.10589495301246643,
                    "Y": 0.3722417950630188
                }
            ],
            "Pose": {
                "Pitch": -0.5589138865470886,
                "Roll": -5.1093974113464355,
                "Yaw": 18.69594955444336
            },
            "Quality": {
                "Brightness": 43.052337646484375,
                "Sharpness": 99.68138885498047
            }
        },
        "Timestamp": 0
    },
    {
        "Face": {
            "BoundingBox": {
                "Height": 0.2177777737379074,
                "Left": 0.7593749761581421,
                "Top": 0.13333334028720856,
                "Width": 0.12250000238418579
            },
            "Confidence": 99.63436889648438,
            "Landmarks": [
                {
                    "Type": "eyeLeft",
                    "X": 0.8005779385566711,
                    "Y": 0.20915353298187256
                },
                {
                    "Type": "eyeRight",
                    "X": 0.8391435146331787,
                    "Y": 0.21049551665782928
                },
                {
                    "Type": "nose",
                    "X": 0.8191410899162292,
                    "Y": 0.2523227035999298
                },
                {
```

```
                        "Type": "mouthLeft",
                        "X": 0.8093273043632507,
                        "Y": 0.29053622484207153
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.8366993069648743,
                        "Y": 0.29101791977882385
                    }
                ],
                "Pose": {
                    "Pitch": 3.165884017944336,
                    "Roll": 1.4182015657424927,
                    "Yaw": -11.151537895202637
                },
                "Quality": {
                    "Brightness": 28.910892486572266,
                    "Sharpness": 97.61507415771484
                }
            },
            "Timestamp": 0
        }.......

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

在集合中搜尋人臉

Amazon Rekognition 可讓您使用輸入人臉來搜尋已儲存人臉集合中的相符人臉。首先，將偵測到的人臉相關資訊儲存在伺服器端容器 (稱為「集合」) 中。集合會儲存個別臉孔和使用者 (同一個人的多張臉孔)。將人臉會儲存為人臉向量，是人臉的數學表現法 (而非人臉的實際映像)。同一個人的不同圖像可用於在同一個集合中創建和存儲多個臉部向量。然後，您可以彙總同一個人的多個臉部向量來創建用戶向量。使用者矢量可以提供更高的人臉搜尋精度，並具有更強大的描述，包含不同程度的照明效果、銳利度、姿勢與外觀等。

建立集合後，您可以使用輸入人臉來搜尋集合中相符的使用者向量或人臉向量。與搜尋個別人臉向量相比，基於使用者向量進行搜尋可大幅提高準確度。您可以使用在映像、存放的影片和串流影片中偵測人臉，來搜尋已儲存的人臉向量。您可以使用在映像中偵測到的人臉來搜尋儲存的使用者向量。

若要存放人臉資訊，您需要執行以下操作：

1. 建立集合-若要儲存臉部資訊，您必須先在帳戶的其中一個 AWS 區域建立 (CreateCollection) 臉部集合。當您呼叫 `IndexFaces` 操作時可指定此人臉集合。
2. 索引臉孔-此IndexFaces作業會偵測影像中的臉部，擷取臉部向量，並將臉部向量儲存在集合中。您可以使用此操作來偵測映像中人臉，並將偵測到的人臉特徵的資訊儲存於集合中。此為以儲存體為基礎的 API 操作範例，因為該操作會在伺服器中儲存資訊。

若要建立使用者並將多個人臉向量與使用者建立關聯，您需要執行下列操作：

1. 建立使用者-您必須先使用建立使用者CreateUser。您可以將同一人物的多個人臉向量彙總至使用者向量，藉此提高人臉配對的精確度。您最多可以將 100 個人臉向量與一個使用者向量建立關聯。
2. 關聯面-建立使用者之後，您可以透過AssociateFaces作業將現有的面向量新增至該使用者。人臉向量必須與使用者向量位於相同的集合中，才能與該使用者向量相關聯。

建立集合並儲存人臉和使用者向量之後，可以使用下列操作來搜尋人臉相符專案：

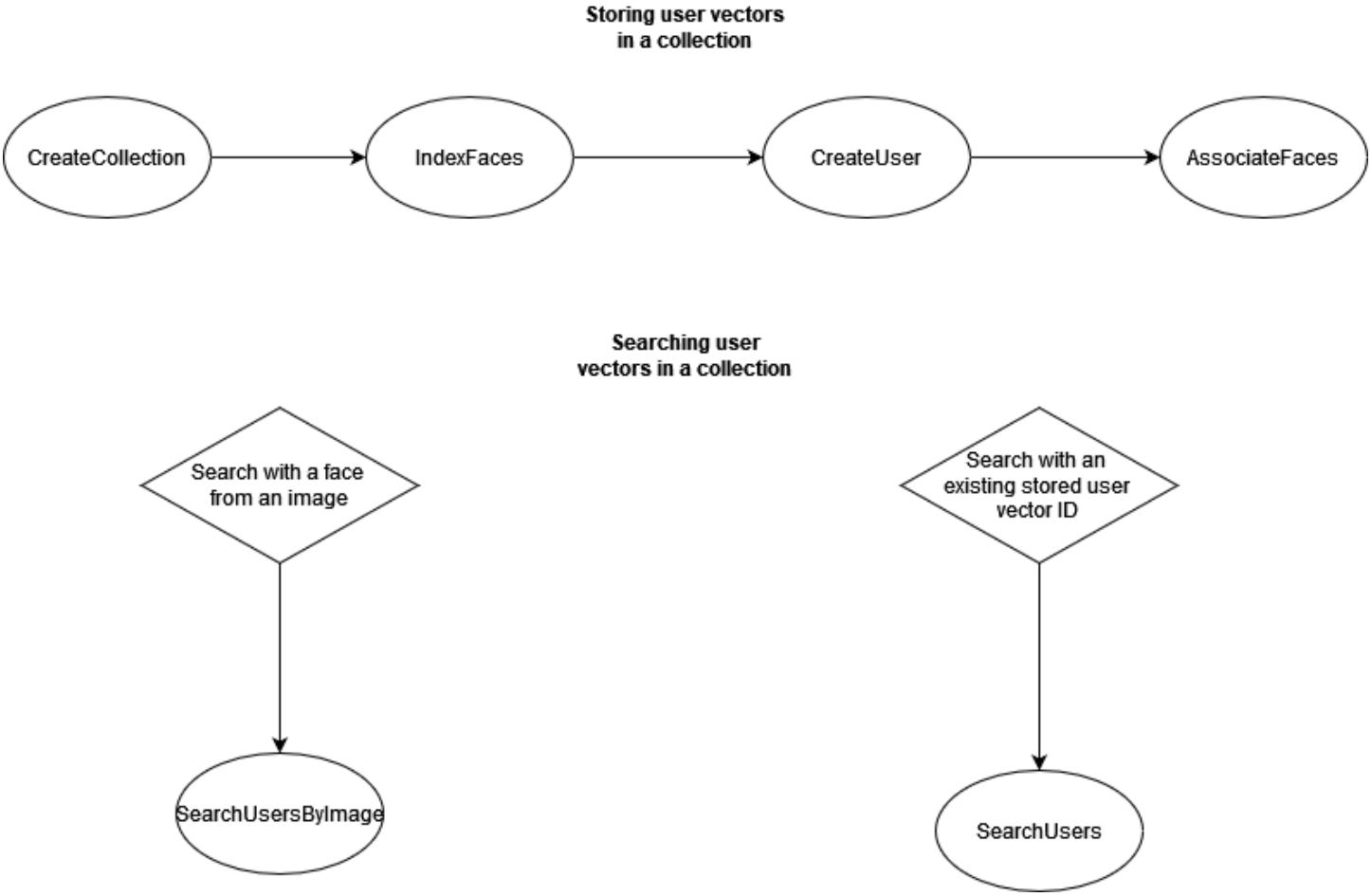
- SearchFacesByImage-從圖像中搜索存儲的帶有臉部的單個面孔。
- SearchFaces-使用提供的面部 ID 搜索存儲的單個面孔。
- SearchUsers-使用提供的面部 ID 或用戶 ID 搜索存儲的用戶。
- SearchUsersByImage-使用圖像中的臉部搜索存儲的用戶。
- StartFaceSearch-在存儲的視頻中搜索面孔。

- CreateStreamProcessor-在流媒體視頻中搜索臉部。

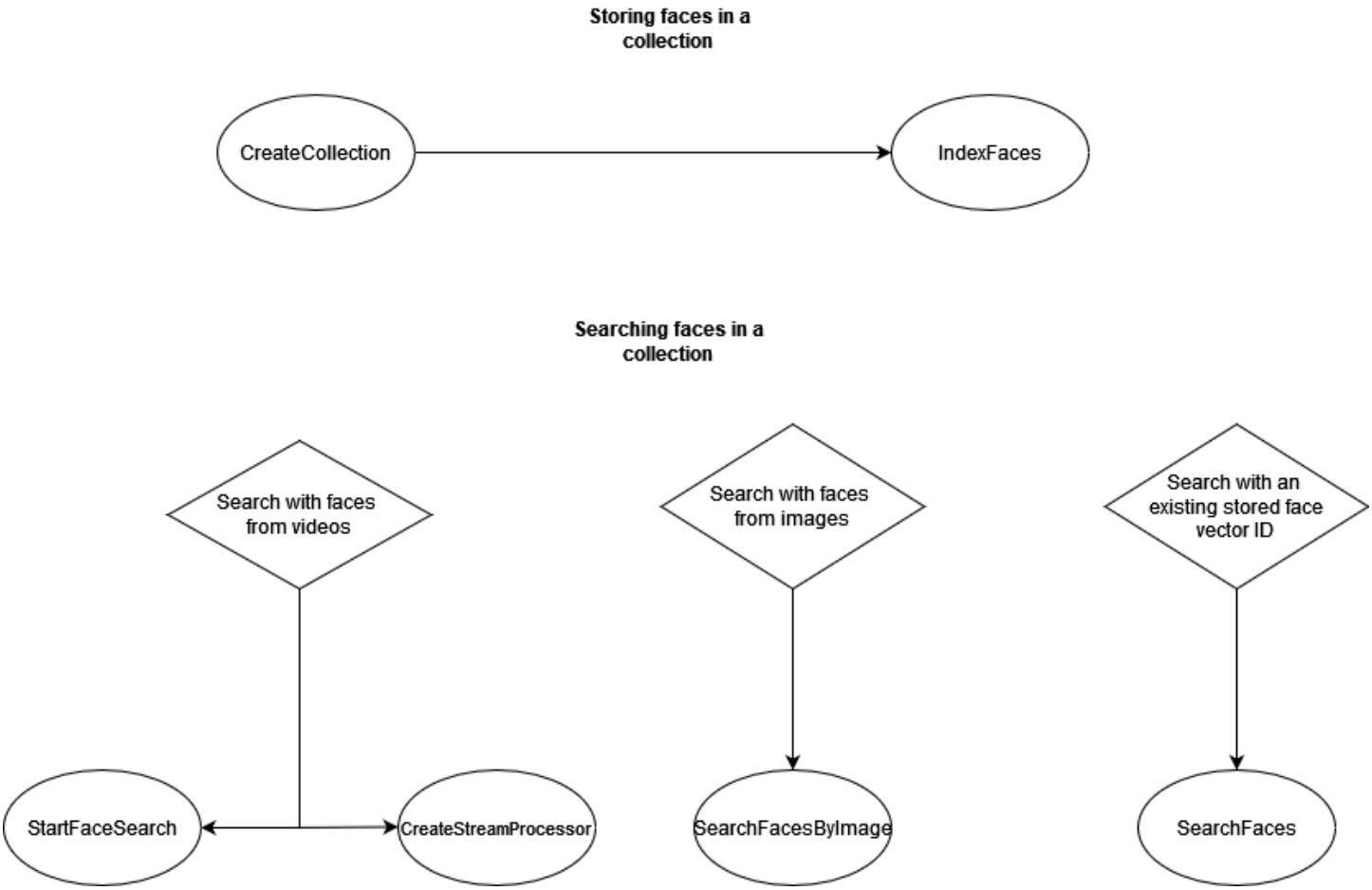
> **Note**
>
> 集合存儲面向量，這是面的數學表示法。集合不會儲存臉部影像。

下圖會根據您使用集合的目標，顯示呼叫作業的順序：

要獲得與用戶向量匹配的最大精度：



對於與個別臉部向量進行高精度匹配：



您可以在各種不同情況中使用集合。例如，您可以建立人臉集合，該集合透過使用 IndexFaces 和 AssociateFaces 操作來儲存從掃描員工徽章映像和政府核發的 ID 中偵測到的人臉。當員工進入大樓時，將拍攝員工人臉的映像並傳送至 SearchUsersByImage 操作。如果臉部比對的相似性分數達到設定的目標 (例如 99%)，即可驗證該員工身分。

管理集合

人臉集合是主要的 Amazon Rekognition 資源，每個您所建立的人臉集合都有一組專有的 Amazon Resource Name (ARN)。您可以在帳戶中的特定 AWS 區域中建立每個臉部集合。當建立集合時，將採用最新版的臉部偵測模型版本。如需詳細資訊，請參閱 模型版本控制。

您可以對集合執行以下管理操作。

- 用 CreateCollection 建立一個集合。如需詳細資訊，請參閱 建立集合。
- 用 ListCollections 列出可用的集合。如需詳細資訊，請參閱 列出集合。
- 用 DescribeCollection 描述一個集合。如需詳細資訊，請參閱 描述集合。

- 用 DeleteCollection 刪除一個集合。如需詳細資訊，請參閱 刪除集合。

管理集合中的人臉

在您建立人臉集合後，可將人臉存入其中。Amazon Rekognition 將提供以下操作，供您管理集合中的人臉。

- 此IndexFaces作業會偵測輸入影像 (JPEG 或 PNG) 中的臉孔，並將其新增至指定的臉部集合。對於在映像中偵測到的每個人臉傳回唯一的人臉 ID。存入臉部之後，便可以搜尋臉部集合來比對臉部。如需詳細資訊，請參閱 新增人臉到集合。
- 此ListFaces作業會列示集合中的面。如需詳細資訊，請參閱 新增人臉到集合。
- 此作DeleteFaces業會從集合中刪除面。如需詳細資訊，請參閱 刪除集合中的人臉。

管理集合中的使用者

儲存來自同一個人的多個人臉向量後，您可以將所有這些人臉向量關聯到一個使用者向量來提高準確性。您可以使用下列操作來管理使用者：

- CreateUser-作業會使用提供的唯一使用者 ID 在集合中建立新使用者。
- AssociateUsers-在用戶 ID 中添加 1-100 個唯一的面部識別碼。將至少一個人臉 ID 與使用者建立關聯後，可以搜尋收藏中的該使用者相符專案。
- ListUsers-列出集合中的使用者。
- DeleteUsers-使用提供的使用者 ID 從集合中刪除使用者。
- DisassociateFaces-從用戶中刪除一個或多個面部 ID。

使用相似性閾值來關聯人臉

確保與使用者相關聯的人臉都來自同一個人很重要。為提供協助，此 `UserMatchThreshold` 參數可指定與至少包含一個 `FaceID` 的 `UserID` 相關聯的新人人臉所需的最小使用者相符可信度。這有助於確保 `FaceIds` 與右側 `UserID` 相關聯。此值的範圍介於 0-100 之間，預設值為 75。

使用指南 IndexFaces

以下是在常見案例中使用 `IndexFaces` 的指導。

關鍵或公共安全應用

- IndexFaces使用每張影像中僅包含一張臉孔的影像進行呼叫，並將傳回的 Face ID 與影像主體的識別碼建立關聯。
- 您可以在建立索引之DetectFaces前使用，以確認影像中只有一個面。如果偵測到多個人臉，請在檢閱只有一個人臉之後重新提交映像。這可防止不小心製作多個臉部的索引，以及將它們與同一個人建立關聯。

照片共享和社交媒體應用程式

- 應該呼叫 `IndexFaces`，而不限制在使用案例中包含多個人臉的映像，例如家庭相冊。在這類情況下，您需要識別每張照片中的每一個人，並使用該資訊依其中呈現的人員將照片分組。

一般用法

- 對同一個人之不同的映像 (尤其是具有不同人臉屬性 (人臉姿勢、人臉毛髮等等)) 進行索引，建立使用者，將不同人臉與使用者關聯，以改善比對品質。
- 包含檢閱程序，以便可以使用正確的臉部識別符來為失敗的比對製作索引，來提高後續的臉部比對能力。
- 如需映像品質的相關資訊，請參閱 人臉比較輸入映像的建議。

在集合中搜尋人臉與使用者

在建立人臉集合並儲存人臉向量和/或使用者向量後，即可搜尋人臉集合來比對人臉。使用 Amazon Rekognition，您可以在符合的集合中搜尋人臉：

- 提供的臉部 ID (SearchFaces)。如需詳細資訊，請參閱 搜尋具有人臉 ID 的人臉。
- 提供的影像中最大的面 (SearchFacesByImage)。如需詳細資訊，請參閱 使用映像來搜尋人臉。
- 已存放影片中的人臉。如需詳細資訊，請參閱 在儲存的影片中搜尋人臉。
- 串流影片中的人臉。如需詳細資訊，請參閱 使用串流視訊事件。

可使用 `CompareFaces` 操作將原始映像中的人臉與目標映像中的人臉進行比較。這項比對的範圍僅限於在目標映像中所偵測到的人臉。如需詳細資訊，請參閱比較映像中的人臉。

下列清單中看到的各種搜尋操作會將人臉 (通過 FaceId 或輸入映像進行識別) 與指定人臉集合中儲存的所有人臉進行比較：

- SearchFaces
- SearchFacesByImage
- SearchUsers
- SearchUsersByImage

使用相似性臨界值來比對人臉

我們允許您透過提供相似性閾值作為輸入參數來控制所有搜尋作業 (CompareFacesSearchUsers、、、、SearchUsersByImage) 的結果。SearchFacesSearchFacesByImage

FaceMatchThreshold 作為 SearchFaces 與 SearchFacesByImage 的相似性閾值輸入屬性，會根據匹配的人臉相似度來控制傳回結果的數量。UserMatchThreshold 作為 SearchUsers 和 SearchUsersByImage 的相似性閾值屬性，會根據匹配的使用者向量相似度來控制的傳回結果數量。臨界值屬性適 SimilarityThreshold 用於 CompareFaces。

含低於臨界值之 Similarity 回應屬性值的回應不會傳回。請務必利用此閾值校正您的使用案例，因為其可決定您的相符結果包含多少錯誤肯定。這可控制您搜尋結果的回呼 (閾值越低，回呼越高)。

所有機器學習系統是機率性的。您應根據使用案例，在設定適當的相似性閾值時使用判斷。例如，如果您想要建立照片應用程式來辨識樣貌相似度家庭成員，您可以選擇較低的閾值 (例如 80%)。另一方面，對於許多強制執行法律使用案例，建議您使用 99% 或更高的臨界值，以降低意外辨識錯誤。

除 FaceMatchThreshold 與 UserMatchThreshold 外，您還可以使用 Similarity 回應屬性作為降低意外辨識錯誤的方法。例如，您可以選擇使用較低閾值 (例如 80%) 來傳回更多結果。接著您可以使用回應屬性相似性 (相似性百分比)，來縮小選擇範圍並篩選應用程式中適當的回應。同樣地，使用更高的相似度 (如 99% 或更高) 將會降低辨識錯誤的風險。

涉及公共安全的使用案例

除 感應器、輸入映像和影片的最佳實務 和 使用指南 IndexFaces 中列出的建議外，在涉及公共安全的使用案例中部署人臉偵測和比較系統時，您應該使用以下最佳實務。首先，您應該使用 99% 或更高的可信度閾值，以降低錯誤和誤報。其次，您應該包含人工檢閱，以驗證從臉部偵測或比較系統收到的結果，而且若沒有額外的人工檢閱，您不應該根據系統輸出做出決策。人臉偵測和比較系統應做為工具，

用於協助縮小範圍，支持人們迅速檢閱和考慮選項。第三，我們建議您在這些使用案例中使用臉部偵測或比較系統時應該透明，包括儘可能將使用這些系統的情形通知終端使用者和主體、獲得此類使用的同意，以及提供一個機制，讓最終使用者和主體可以提供意見回饋以改善系統。

如果您是執法部門，在進行刑事調查時需用到 Amazon Rekognition 人臉比較特徵，您必須遵守 AWS 服務條款中列出的要求。這包含下列專案：

- 由經過適當培訓的人員審查所有決定，採取可能影響公民自由或同等人權的行動。
- 培訓人員負責使用臉部辨識系統。
- 公開您使用臉部辨識系統的方式。
- 未經獨立審查或若非緊急情況，不得使用 Amazon Rekognition 持續監控一物件。

在所有情況下，應在具有其他強而有力的證據下檢視臉部比較配對，而不應將其做為採取行動的唯一決定因素。但是，如果面部比較用於 non-law-enforcement 案例（例如，用於解鎖電話或驗證員工的身份以訪問安全的私人辦公大樓），則這些決定不需要進行手動審核，因為它們不會影響一個人的公民自由或同等的人權。

如果您打算針對涉及公共安全的使用案例使用臉部偵測或臉部比較系統，則應該採用先前提及的最佳實務。此外，您應該參考已發佈的資源，了解如何使用臉部比較。其中包括司法部司法援助局所提供的 Face Recognition Policy Development Template For Use In Criminal Intelligence and Investigative Activities。此範本提供多種臉部比較和生物相關資源，旨在提供執法和公共安全部門一個框架，以開發遵守適用法律、降低隱私風險，並建立實體責任和監督的臉部比較政策。其他資源包括美國國家電信暨資訊管理局提供的 Best Privacy Practices for Commercial Use of Facial Recognition，以及美國聯邦貿易委員會全體提供的 Best Practices for Common Uses of Facial Recognition。可能會在未來開發並發佈其他資源，而且您應該持續對此重要主題自我教育。

提醒您，您在使用 AWS 服務時必須遵守所有適用的法律，而且若使用的方式違反其他人的權利或可能有害其他人的話，您可能會無法使用任何 AWS 服務。這意味著您不得以非法歧視個人或侵犯個人正當程序、隱私或公民自由的方式將 AWS 服務用於公共安全使用案例。您應該視需要取得適當的法律建議，以查閱關於使用案例的任何法律要求或問題。

使用 Amazon Rekognition 協助公眾安全

Amazon Rekognition 可協助公共安全與執法部門要求案例 (例如尋找失蹤兒童、打擊人口販賣或預防犯罪)。在公共安全與執法部門要求案例中，請考慮以下：

- 使用 Amazon Rekognition 做為尋找可能配對的第一步。透過 Amazon Rekognition 人臉操作的回應可讓您快速地取得潛在配對，供您進一步考量。

- 請勿使用 Amazon Rekognition 回應在需要人類分析的情境中進行自主決策。如果您是執法部門，需使用 Amazon Rekognition 協助識別與刑事調查相關的物件，且所採取的身分類型行動可能影響該人的公民自由或同等人權，負責決定採取行動的人員必須接受過獨立驗證身分證明的相應培訓。
- 將 99% 相似度的臨界值用於需要高準確臉部相似度比對的案例。此類範例即是進入建築物時的存取驗證。
- 當有公民權利這方面的考量時，例如包含執法部門的使用案例，請使用 99% 或更高的臨界值，以及採用人工的方式檢閱臉部比較預測結果，以確保未侵犯人員的公民權利。
- 將低於 99% 相似度臨界值用於受益於大型潛在相符集合的案例中。此類範例其中之一便是尋找失蹤人口。如有需要，您可以使用相似度回應屬性，來判斷與您希望辨識之人員的潛在相符的相似程度。
- 擬訂計畫，來判斷 Amazon Rekognition 所傳回之相符人臉的對錯。例如，當您使用IndexFaces作業建立索引時，使用同一人物的多個影像來改善相符。如需詳細資訊，請參閱 使用指南 IndexFaces。

在其他使用案例 (例如社交媒體)，我們建議您使用最佳判斷，以評估 Amazon Rekognition 結果是否需要人工檢閱。此外，根據應用程式的要求，相似度臨界值可能較低。

建立集合

您可以使用該CreateCollection操作來創建集合。

如需詳細資訊，請參閱 管理集合。

若要建立集合 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 `AmazonRekognitionFullAccess` 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 `CreateCollection` 操作。

   Java

   以下範例建立集合和顯示其 Amazon Resource Name (ARN)。

   將 `collectionId` 的值變更為您要建立集合的名稱。

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

package aws.example.rekognition.image;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CreateCollectionRequest;
import com.amazonaws.services.rekognition.model.CreateCollectionResult;


public class CreateCollection {

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


      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


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

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

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

   }

}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

```
//snippet-start:[rekognition.java2.create_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
//snippet-end:[rekognition.java2.create_collection.import]

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

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <collectionName> \n\n" +
         "Where:\n" +
         "   collectionName - The name of the collection. \n\n";

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

     String collectionId = args[0];
     Region region = Region.US_EAST_1;
```

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

    System.out.println("Creating collection: " +collectionId);
    createMyCollection(rekClient, collectionId );
    rekClient.close();
}

// snippet-start:[rekognition.java2.create_collection.main]
public static void createMyCollection(RekognitionClient rekClient,String
collectionId ) {

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

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

    } catch(RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.create_collection.main]
```

AWS CLI

此 AWS CLI 命令會顯示 `create-collection` CLI 作業的 JSON 輸出。

將 `collection-id` 的值取代為您要建立集合的名稱。

使用您開發人員設定檔的名稱取代 `profile_name` 的值。

```
aws rekognition create-collection --profile profile-name --collection-id
 "collection-name"
```

Python

以下範例建立集合和顯示其 Amazon Resource Name (ARN)。

將 `collection_id` 的值變更為您要建立集合的名稱。將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

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

import boto3

def create_collection(collection_id):
    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

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

def main():
    collection_id = "collection-id"
    create_collection(collection_id)

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

.NET

以下範例建立集合和顯示其 Amazon Resource Name (ARN)。

將 `collectionId` 的值變更為您要建立集合的名稱。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

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

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

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

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

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

    }
}
```

Node.JS

在下列範例中，將 region 的值取代為與您帳戶相關聯的區域名稱，並將 collectionName 的值替代為您集合的所需名稱。

將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

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

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const collectionName = "collection-name"
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

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

 createCollection(collectionName)
```

CreateCollection 操作請求

CreationCollection 的輸入是您要建立的集合的名稱。

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

CreateCollection 作業回應

Amazon Rekognition 建立集合並傳回新建立的集合之 Amazon Resource Name (ARN)。

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

標記集合

您可以使用標籤來識別、組織、搜尋和篩選 Amazon Rekognition 集合。每個標籤都是由使用者定義的金鑰和值組成的標籤。

您也能通過 Identity and Access Management (IAM) 使用標籤來管控集合的存取。如需詳細資訊，請參閱使用 AWS 資源標籤控制資源的存取。

主題

- 將標籤新增至新集合
- 將標籤新增至現有集合
- 列出集合中的標籤
- 刪除集合中的標籤

將標籤新增至新集合

您可以在使用 `CreateCollection` 操作建立集合時，將標籤新增至集合。在 `Tags` 陣列輸入參數中指定一或多個標籤。

AWS CLI

使用您開發人員設定檔的名稱取代 `profile_name` 的值。

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

對於 Windows 裝置：

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

Python

將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

```
import boto3

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

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

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

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

將標籤新增至現有集合

若要將一或多個標籤新增至現有的線索，請使用 `TagResource` 操作。指定待新增的集合的 Amazon Resource Name (ARN) (`ResourceArn`) 和標籤 (`Tags`)。下列範例顯示如何結合標記。

AWS CLI

使用您開發人員設定檔的名稱取代 `profile_name` 的值。

```
aws rekognition tag-resource --resource-arn collection-arn --tags
 "{"key1":"value1","key2":"value2"}" --profile profile-name
```

對於 Windows 裝置：

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

Python

將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

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

import boto3

def create_tag(collection_id):
    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')
    response = client.tag_resource(ResourceArn=collection_id,
                                   Tags={
                                       "KeyName": "ValueName"
                                   })
    print(response)
    if "'HTTPStatusCode': 200" in str(response):
        print("Success!!")

def main():
    collection_arn = "collection-arn"
    create_tag(collection_arn)

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

> **Note**
>
> 如果您不知道集合的 Amazon Resource Name，則可以使用該 `DescribeCollection` 操作。

列出集合中的標籤

若要列出附加至集合的標籤，請使用 ListTagsForResource 操作並指定集合的 ARN (ResourceArn)。回應是連接到指定集合的標籤鍵和值的映射。

AWS CLI

使用您開發人員設定檔的名稱取代 profile_name 的值。

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

Python

將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
import boto3

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

def main():
    list_tags()

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

輸出會顯示連接至集合的標籤清單：

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

刪除集合中的標籤

若要移除集合中的一個或多個標籤，請使用此 UntagResource 操作。指定要移除的模型 (ResourceArn) 和標籤鍵 (Tag-Keys) 的 ARN。

AWS CLI

使用您開發人員設定檔的名稱取代 profile_name 的值。

```
aws rekognition untag-resource --resource-arn resource-arn --profile profile-name --
tag-keys "key1" "key2"
```

或者，您可以使用以下格式指定標籤鍵：

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

Python

將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
import boto3

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

def main():
    list_tags()

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

列出集合

您可以使用此ListCollections作業列出所使用區域中的集合。

如需詳細資訊，請參閱 管理集合。

若要列出集合 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 ListCollections 操作。

   Java

   以下範例列出目前區域中的集合。

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

package aws.example.rekognition.image;

import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.ListCollectionsRequest;
import com.amazonaws.services.rekognition.model.ListCollectionsResult;

public class ListCollections {

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


      AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();


      System.out.println("Listing collections");
      int limit = 10;
      ListCollectionsResult listCollectionsResult = null;
      String paginationToken = null;
      do {
         if (listCollectionsResult != null) {
```

```
            paginationToken = listCollectionsResult.getNextToken();
         }
         ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
                 .withMaxResults(limit)
                 .withNextToken(paginationToken);

 listCollectionsResult=amazonRekognition.listCollections(listCollectionsRequest);

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

   }
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.list_collections.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsRequest;
import
 software.amazon.awssdk.services.rekognition.model.ListCollectionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
//snippet-end:[rekognition.java2.list_collections.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
```

```
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class ListCollections {

 public static void main(String[] args) {

     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     System.out.println("Listing collections");
     listAllCollections(rekClient);
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.list_collections.main]
 public static void listAllCollections(RekognitionClient rekClient) {
     try {
         ListCollectionsRequest listCollectionsRequest =
 ListCollectionsRequest.builder()
             .maxResults(10)
             .build();

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

     } catch (RekognitionException e) {
         System.out.println(e.getMessage());
         System.exit(1);
     }
 }
 // snippet-end:[rekognition.java2.list_collections.main]
}
```

AWS CLI

此 AWS CLI 命令會顯示 list-collections CLI 作業的 JSON 輸出。使用您開發人員設定檔的名稱取代 profile_name 的值。

```
aws rekognition list-collections --profile profile-name
```

Python

以下範例列出目前區域中的集合。

將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3

def list_collections():

    max_results=2

    client=boto3.client('rekognition')

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

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

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

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

```
        else:
            done=True

    return collection_count

def main():

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

.NET

以下範例列出目前區域中的集合。

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

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

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

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

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

            ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
```

```
            {
                MaxResults = limit,
                NextToken = paginationToken
            };

            listCollectionsResponse =
 rekognitionClient.ListCollections(listCollectionsRequest);

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

Node.js

將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

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

import { ListCollectionsCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

const listCollection = async () => {
  var max_results = 10
  console.log("Displaying collections:")
  var response = await rekogClient.send(new ListCollectionsCommand({MaxResults:
 max_results}))
  var collection_count = 0
  var done = false
```

```
  while (done == false){
      var collections = response.CollectionIds
      collections.forEach(collection => {
          console.log(collection)
          collection_count += 1
      });
      return collection_count
  }
}

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

ListCollections 操作請求

ListCollections 的輸入是要傳回的最大集合數。

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

如果回應的集合多於 MaxResults 請求，則會傳回一個字符，您可以在後續的 ListCollections 呼叫中使用該字符獲取下一組結果。例如：

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

ListCollections 作業回應

Amazon Rekognition 傳回集合陣列 (CollectionIds)。個別陣列 (FaceModelVersions) 提供用於分析每個集合中人臉的人臉模型的版本。例如，在下列 JSON 回應中，集合 MyCollection 使用臉部模型的 2.0 版分析臉部。集合 AnotherCollection 使用臉部模型的 3.0 版本。如需詳細資訊，請參閱 模型版本控制。

NextToken 是在後續的 ListCollections 呼叫中用於取得下一組結果的字符。

```
{
    "CollectionIds": [
```

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

描述集合

您可以使用該DescribeCollection操作來獲取有關集合的以下信息：

- 編制為集合索引的臉部數。
- 與集合搭配使用的模型版本。如需詳細資訊，請參閱 the section called “模型版本控制”。
- 集合的 Amazon Resource Name (ARN)。
- 集合的建立日期和時間。

描述集合 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 DescribeCollection 操作。

   Java

   此範例描述一個集合。

   將 collectionId 的值變更為所需集合的 ID。

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

```
package com.amazonaws.samples;

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


public class DescribeCollection {

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

      String collectionId = "CollectionID";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


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


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

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

   }

}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
//snippet-end:[rekognition.java2.describe_collection.import]

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

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <collectionName>\n\n" +
         "Where:\n" +
         "   collectionName - The name of the Amazon Rekognition collection. \n
\n";

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

     String collectionName = args[0];
     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
```

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

        describeColl(rekClient, collectionName);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.describe_collection.main]
    public static void describeColl(RekognitionClient rekClient, String
    collectionName) {

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

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

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.describe_collection.main]
}
```

AWS CLI

此 AWS CLI 命令會顯示 `describe-collection` CLI 作業的 JSON 輸出。將 `collection-id` 的值變更為所需集合的 ID。將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

```
aws rekognition describe-collection --collection-id collection-name --profile
 profile-name
```

Python

此範例描述一個集合。

將 collection_id 的值變更為所需集合的 ID。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3
from botocore.exceptions import ClientError

def describe_collection(collection_id):

    print('Attempting to describe collection ' + collection_id)

    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')

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

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

def main():
    collection_id = 'collection-name'
    describe_collection(collection_id)

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

.NET

此範例描述一個集合。

將 collectionId 的值變更為所需集合的 ID。

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

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

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

        String collectionId = "CollectionID";
        Console.WriteLine("Describing collection: " + collectionId);

        DescribeCollectionRequest describeCollectionRequest = new
 DescribeCollectionRequest()
        {
            CollectionId = collectionId
        };

        DescribeCollectionResponse describeCollectionResponse =
 rekognitionClient.DescribeCollection(describeCollectionRequest);
        Console.WriteLine("Collection ARN: " +
 describeCollectionResponse.CollectionARN);
        Console.WriteLine("Face count: " +
 describeCollectionResponse.FaceCount);
        Console.WriteLine("Face model version: " +
 describeCollectionResponse.FaceModelVersion);
        Console.WriteLine("Created: " +
 describeCollectionResponse.CreationTimestamp);
    }
```

```
}
```

Node.js

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

import { DescribeCollectionCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// Name the collection
const collection_name = "collection-name"

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

```
describeCollection(collection_name)
```

DescribeCollection 操作請求

DescribeCollection 的輸入是所需集合的 ID，如以下 JSON 範例所示。

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

DescribeCollection作業回應

回應包括：

- 編制為集合索引的臉部數，FaceCount。
- 與集合配合使用的面模型版本 FaceModelVersion。如需詳細資訊，請參閱 the section called “模型版本控制”。
- 集合 Amazon Resource Name，CollectionARN。
- 集合的建立日期和時間，CreationTimestamp。CreationTimestamp 的值是 Unix epoch 時間到集合建立時所經的毫秒數。Unix epoch 時間為 1970 年 1 月 1 日周四 00:00:00 國際標準時間 (UTC)。如需詳細資訊，請參閱 Unix 時間。

```
{
    "CollectionARN": "arn:aws:rekognition:us-east-1:nnnnnnnnnnnn:collection/
MyCollection",
    "CreationTimestamp": 1.533422155042E9,
    "FaceCount": 200,
    "UserCount" : 20,
    "FaceModelVersion": "1.0"
}
```

刪除集合

您可以使用此DeleteCollection作業刪除集合。

如需詳細資訊，請參閱 管理集合。

若要刪除集合 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 DeleteCollection 操作。

   Java

   此範例刪除一個集合。

   將值 collectionId 變更為您要刪除的集合。

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

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteCollectionRequest;
import com.amazonaws.services.rekognition.model.DeleteCollectionResult;


public class DeleteCollection {

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

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      String collectionId = "MyCollection";

      System.out.println("Deleting collections");

      DeleteCollectionRequest request = new DeleteCollectionRequest()
         .withCollectionId(collectionId);
      DeleteCollectionResult deleteCollectionResult =
 rekognitionClient.deleteCollection(request);
```

```
      System.out.println(collectionId + ": " +
 deleteCollectionResult.getStatusCode()
         .toString());

   }

}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
// snippet-start:[rekognition.java2.delete_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
// snippet-end:[rekognition.java2.delete_collection.import]

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

    public static void main(String[] args) {

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

```
            "   <collectionId> \n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection to delete. \n\n";

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

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.delete_collection.main]
    public static void deleteMyCollection(RekognitionClient rekClient,String
 collectionId ) {

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

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

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.delete_collection.main]
```

```
}
```

AWS CLI

此 AWS CLI 命令會顯示 delete-collection CLI 作業的 JSON 輸出。將 collection-id 的值取代為您要刪除的集合的名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
aws rekognition delete-collection --collection-id collection-name --profile
 profile-name
```

Python

此範例刪除一個集合。

將值 collection_id 變更為您要刪除的集合。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3
from botocore.exceptions import ClientError

def delete_collection(collection_id):

    print('Attempting to delete collection ' + collection_id)
    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')

    status_code = 0

    try:
        response = client.delete_collection(CollectionId=collection_id)
        status_code = response['StatusCode']

    except ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFoundException':
            print('The collection ' + collection_id + ' was not found ')
        else:
```

```
            print('Error other than Not Found occurred: ' + e.response['Error']
['Message'])
        status_code = e.response['ResponseMetadata']['HTTPStatusCode']
    return (status_code)

def main():

    collection_id = 'collection-name'
    status_code = delete_collection(collection_id)
    print('Status code: ' + str(status_code))

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

.NET

此範例刪除一個集合。

將值 collectionId 變更為您要刪除的集合。

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

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

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

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

        DeleteCollectionRequest deleteCollectionRequest = new
 DeleteCollectionRequest()
        {
            CollectionId = collectionId
```

```
        };

        DeleteCollectionResponse deleteCollectionResponse =
 rekognitionClient.DeleteCollection(deleteCollectionRequest);
        Console.WriteLine(collectionId + ": " +
 deleteCollectionResponse.StatusCode);
    }
}
```

Node.js

將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

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

import { DeleteCollectionCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// Name the collection
const collection_name = "collection-name"

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

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

deleteCollection(collection_name)
```

DeleteCollection 操作請求

`DeleteCollection` 的輸入是要刪除的集合 ID，如以下 JSON 範例所示。

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

DeleteCollection 作業回應

`DeleteCollection` 回應包含指示操作成功或失敗的 HTTP 狀態碼的。若集合成功刪除則傳回 `200`。

```
{"StatusCode":200}
```

新增人臉到集合

您可以使用此IndexFaces作業偵測影像中的臉孔並將其加入集合中。對於每個偵測到的人臉，Amazon Rekognition 將擷取人臉特徵並將特徵資訊存放於資料庫中。此外，該命令將為每個在指定人臉集合中偵測到的人臉儲存中繼資料。Amazon Rekognition 不會儲存實際的映像位元組。

如需為編制索引提供適合臉部的相關資訊，請參閱 人臉比較輸入映像的建議。

`IndexFaces` 操作將保留每個人臉的下列資訊：

- 多維度人臉特徵：`IndexFaces` 使用人臉分析來擷取人臉特徵的多維度資訊，並將資訊儲存於人臉集合中。您無法直接存取此資訊。不過，Amazon Rekognition 會在搜尋人臉集合並比對人臉時使用此資訊。

- 中繼資料：每個人臉的中繼資料包括要求的週框方塊、可信度等級 (週框方塊標註的人臉)、由 Amazon Rekognition 指定的 ID (人臉 ID 與映像 ID)、以及請求中的外部映像 ID (若您提供此資訊)。

此資訊會在對於 IndexFaces API 呼叫的回應中回傳給您。如需範例，請參閱下列範例回應中的 face 元素。

此服務會傳回此中繼資料，以回應下列 API 呼叫：

- ListFaces
- 搜尋臉孔操作 — 回應SearchFaces並SearchFacesByImage傳回每個相符臉孔的相符項目的信賴度，以及相符臉孔的這個中繼資料。

由 IndexFaces 索引的臉部數量取決於與輸入集合相關聯的臉部偵測模型的版本。如需詳細資訊，請參閱 模型版本控制。

有關索引面的資訊會傳回FaceRecord物件陣列中。

您可能想要將編入索引的人臉與偵測到這些人臉的映像連結。例如，您可能想要維護映像的使用者端索引以及映像中的人臉。若要將人臉與映像連結，請在 ExternalImageId 請求參數中指定一個映像 ID。映像 ID 可以是檔案名稱或建立的另一個 ID。

除了上述關於 API 保留於人臉集合中的資訊外，API 也會傳回未保留於集合中的人臉詳細資訊。(請參閱下列範例回應中的 faceDetail 元素)。

> **Note**
>
> DetectFaces 將傳回相同的資訊，因此您不需要同時呼叫 DetectFaces 和 IndexFaces。

篩選人臉

此 IndexFaces 作業可讓您篩選從影像建立索引的臉孔。運用 IndexFaces，您可以指定索引建立時所要使用的最多人臉數目，或者選擇只對高品質偵測人臉建立索引。

使用 MaxFaces 輸入參數，您可以指定 IndexFaces 在建立索引時所用的最多人臉數目。若您在建立索引時想要運用映像中尺寸最大的人臉，而不要使用尺寸最小的人臉時，例如，背景中站立人群的人臉，就能使用這項參數。

在預設情況下，IndexFaces 選擇用於篩選出臉部的品質列。您可以使用 QualityFilter 輸入參數，以明確設定品質列。這些值為：

- AUTO：Amazon Rekognition 選擇並不用於篩選出人臉的品質列 (預設值)。
- LOW：除了品質最低的人臉之外，所有人臉都會加以索引。
- MEDIUM
- HIGH：僅對品質最高的人臉行索引。
- NONE：不會根據品質來篩選掉任何人臉。

IndexFaces 篩選人臉的原因如下：

- 相較於映像尺寸，人臉過小。
- 臉部過於模糊。
- 映像過暗。
- 臉部表情過於誇張。
- 臉部沒有足夠的細節，不適合臉部搜尋。

> **Note**
>
> 若要使用品質篩選，您必須使用臉部模型第 3 版或更高版本的相關聯集合。若要取得與集合關聯的臉部模型版本，請呼叫DescribeCollection。

未建立索引之臉孔的相關資訊會傳回UnindexedFace物件陣列中。IndexFacesReasons 陣列包含為何無法為人臉建立索引的原因清單。例如，當值為 EXCEEDS_MAX_FACES 時，表示因為已偵測到透過 MaxFaces 指定的人臉數目，所以無法為人臉建立索引。

如需詳細資訊，請參閱 管理集合中的人臉。

若要新增臉部到集合 (SDK)

1. 如果您尚未執行：
   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。
   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 將映像 (含有一個或多個人臉) 上傳至您的 Amazon S3 儲存貯體。

如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例來呼叫 IndexFaces 操作。

Java

此範例顯示新增到集合的人臉的人臉識別符。

將 collectionId 的值變更為要新增到人臉的集合的名稱。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。.withMaxFaces(1) 參數會將用於建立索引的人臉數目限制為 1。移除或變更其值以符合您的需求。

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

package aws.example.rekognition.image;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.FaceRecord;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.IndexFacesRequest;
import com.amazonaws.services.rekognition.model.IndexFacesResult;
import com.amazonaws.services.rekognition.model.QualityFilter;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.UnindexedFace;
import java.util.List;

public class AddFacesToCollection {
    public static final String collectionId = "MyCollection";
    public static final String bucket = "bucket";
    public static final String photo = "input.jpg";

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        Image image = new Image()
                .withS3Object(new S3Object()
                .withBucket(bucket)
```

```
                .withName(photo));

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
                .withImage(image)
                .withQualityFilter(QualityFilter.AUTO)
                .withMaxFaces(1)
                .withCollectionId(collectionId)
                .withExternalImageId(photo)
                .withDetectionAttributes("DEFAULT");

        IndexFacesResult indexFacesResult =
 rekognitionClient.indexFaces(indexFacesRequest);

        System.out.println("Results for " + photo);
        System.out.println("Faces indexed:");
        List<FaceRecord> faceRecords = indexFacesResult.getFaceRecords();
        for (FaceRecord faceRecord : faceRecords) {
            System.out.println("  Face ID: " +
 faceRecord.getFace().getFaceId());
            System.out.println("  Location:" +
 faceRecord.getFaceDetail().getBoundingBox().toString());
        }

        List<UnindexedFace> unindexedFaces =
 indexFacesResult.getUnindexedFaces();
        System.out.println("Faces not indexed:");
        for (UnindexedFace unindexedFace : unindexedFaces) {
            System.out.println("  Location:" +
 unindexedFace.getFaceDetail().getBoundingBox().toString());
            System.out.println("  Reasons:");
            for (String reason : unindexedFace.getReasons()) {
                System.out.println("   " + reason);
            }
        }
    }
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.add_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
```

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.IndexFacesResponse;
import software.amazon.awssdk.services.rekognition.model.IndexFacesRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.QualityFilter;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceRecord;
import software.amazon.awssdk.services.rekognition.model.UnindexedFace;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Reason;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.add_faces_collection.import]

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

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "    <collectionId> <sourceImage>\n\n" +
         "Where:\n" +
         "    collectionName - The name of the collection.\n" +
         "    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

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

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

    addToCollection(rekClient, collectionId, sourceImage);
    rekClient.close();
}

// snippet-start:[rekognition.java2.add_faces_collection.main]
public static void addToCollection(RekognitionClient rekClient, String
collectionId, String sourceImage) {

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

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

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

        List<UnindexedFace> unindexedFaces = facesResponse.unindexedFaces();
        System.out.println("Faces not indexed:");
        for (UnindexedFace unindexedFace : unindexedFaces) {
```

```
            System.out.println("  Location:" +
 unindexedFace.faceDetail().boundingBox().toString());
            System.out.println("  Reasons:");
            for (Reason reason : unindexedFace.reasons()) {
                System.out.println("Reason:  " + reason);
            }
        }

    } catch (RekognitionException | FileNotFoundException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.add_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令會顯示 index-faces CLI 作業的 JSON 輸出。

以您要儲存臉部的集合之名稱取代 collection-id 的值。將 Bucket 與 Name 的值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像檔案名稱。max-faces 參數會將用於建立索引的人臉數目限制為 1。移除或變更其值以符合您的需求。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
aws rekognition index-faces --image '{"S3Object":{"Bucket":"bucket-
name","Name":"file-name"}}' --collection-id "collection-id" \
                                --max-faces 1 --quality-filter "AUTO" --
detection-attributes "ALL" \
                                --external-image-id "example-image.jpg" --
profile profile-name
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
aws rekognition index-faces --image "{\"S3Object\":{\"Bucket\":\"bucket-name\",
\"Name\":\"image-name\"}}" \
--collection-id "collection-id" --max-faces 1 --quality-filter "AUTO" --
detection-attributes "ALL" \
--external-image-id "example-image.jpg" --profile profile-name
```

Python

此範例顯示新增到集合的人臉的人臉識別符。

將 collectionId 的值變更為要新增到人臉的集合的名稱。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。MaxFaces 輸入參數會將用於建立索引的人臉數目限制為 1。移除或變更其值以符合您的需求。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3

def add_faces_to_collection(bucket, photo, collection_id):

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

    response = client.index_faces(CollectionId=collection_id,
                                  Image={'S3Object': {'Bucket': bucket, 'Name':
 photo}},
                                  ExternalImageId=photo,
                                  MaxFaces=1,
                                  QualityFilter="AUTO",
                                  DetectionAttributes=['ALL'])

    print('Results for ' + photo)
    print('Faces indexed:')
    for faceRecord in response['FaceRecords']:
        print('  Face ID: ' + faceRecord['Face']['FaceId'])
        print('  Location: {}'.format(faceRecord['Face']['BoundingBox']))

    print('Faces not indexed:')
    for unindexedFace in response['UnindexedFaces']:
        print(' Location: {}'.format(unindexedFace['FaceDetail']
['BoundingBox']))
        print(' Reasons:')
        for reason in unindexedFace['Reasons']:
            print('   ' + reason)
    return len(response['FaceRecords'])
```

```
def main():
    bucket = 'bucket-name'
    collection_id = 'collection-id'
    photo = 'photo-name'

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

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

.NET

此範例顯示新增到集合的人臉的人臉識別符。

將 collectionId 的值變更為要新增到人臉的集合的名稱。將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 Amazon S3 儲存貯體名稱與映像名稱。

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        Image image = new Image()
        {
            S3Object = new S3Object()
            {
```

```
                Bucket = bucket,
                Name = photo
            }
        };

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
        {
            Image = image,
            CollectionId = collectionId,
            ExternalImageId = photo,
            DetectionAttributes = new List<String>(){ "ALL" }
        };

        IndexFacesResponse indexFacesResponse =
 rekognitionClient.IndexFaces(indexFacesRequest);

        Console.WriteLine(photo + " added");
        foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
            Console.WriteLine("Face detected: Faceid is " +
               faceRecord.Face.FaceId);
    }
}
```

IndexFaces 操作請求

IndexFaces 的輸入是要索引的映像和新增人臉或人臉的集合。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "ExternalImageId": "input.jpg",
    "DetectionAttributes": [
        "DEFAULT"
    ],
    "MaxFaces": 1,
    "QualityFilter": "AUTO"
}
```

IndexFaces 作業回應

IndexFaces 傳回有關在映像中偵測到的人臉的資訊。例如，以下 JSON 回應包含輸入映像中偵測到的人臉的預設偵測屬性。範例當中還會示範已經超過 MaxFaces 輸入參數的值 (Reasons 陣列包含 EXCEEDS_MAX_FACES)，導致無法為人臉建立索引。當 Reasons 包含像是 LOW_SHARPNESS 或 LOW_BRIGHTNESS 等品質原因時，導致無法為人臉建立索引。如需詳細資訊，請參閱UnindexedFace。


```
{
    "FaceModelVersion": "3.0",
    "FaceRecords": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.3247932195663452,
                    "Left": 0.5055555701255798,
                    "Top": 0.2743072211742401,
                    "Width": 0.21444444358348846
                },
                "Confidence": 99.99998474121094,
                "ExternalImageId": "input.jpg",
                "FaceId": "b86e2392-9da1-459b-af68-49118dc16f87",
                "ImageId": "09f43d92-02b6-5cea-8fbd-9f187db2050d"
            },
            "FaceDetail": {
                "BoundingBox": {
                    "Height": 0.3247932195663452,
                    "Left": 0.5055555701255798,
                    "Top": 0.2743072211742401,
                    "Width": 0.21444444358348846
                },
                "Confidence": 99.99998474121094,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.5751981735229492,
                        "Y": 0.4010535478591919
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.6511467099189758,
                        "Y": 0.4017036259174347
                    },
```

```
                    {
                        "Type": "nose",
                        "X": 0.6314528584480286,
                        "Y": 0.4710812568664551
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.5879443287849426,
                        "Y": 0.5171778798103333
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.6444502472877502,
                        "Y": 0.5164633989334106
                    }
                ],
                "Pose": {
                    "Pitch": -10.313642501831055,
                    "Roll": -1.0316886901855469,
                    "Yaw": 18.079818725585938
                },
                "Quality": {
                    "Brightness": 71.2919921875,
                    "Sharpness": 78.74752044677734
                }
            }
        }
    ],
    "OrientationCorrection": "",
    "UnindexedFaces": [
        {
            "FaceDetail": {
                "BoundingBox": {
                    "Height": 0.1329464465379715,
                    "Left": 0.5611110925674438,
                    "Top": 0.6832437515258789,
                    "Width": 0.08777777850627899
                },
                "Confidence": 92.37225341796875,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.5796897411346436,
                        "Y": 0.7452847957611084
```

```
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.6078574657440186,
                        "Y": 0.742687463760376
                    },
                    {
                        "Type": "nose",
                        "X": 0.597953200340271,
                        "Y": 0.7620673179626465
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.5884202122688293,
                        "Y": 0.7920381426811218
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.60627681016922,
                        "Y": 0.7919750809669495
                    }
                ],
                "Pose": {
                    "Pitch": 15.658954620361328,
                    "Roll": -4.583454608917236,
                    "Yaw": 10.558992385864258
                },
                "Quality": {
                    "Brightness": 42.54612350463867,
                    "Sharpness": 86.93206024169922
                }
            },
            "Reasons": [
                "EXCEEDS_MAX_FACES"
            ]
        }
    ]
}
```

若要取得所有臉部資訊，請指定 DetectionAttributes 請求參數為『ALL』。例如，在以下範例回應中可找到 faceDetail 元素中的其它資訊，而這些資訊不會儲存在伺服器上：

- 25 個臉部特徵點 (相較於上述範例中只有 5 個特徵點)

- 十個面部屬性 (眼鏡、鬍鬚、人臉遮挡、眼睛凝視方向等)
- 表情 (請參閱 emotion 元素)

face 元素提供保留在伺服器上的中繼資料。

FaceModelVersion 是與集合相關聯的臉部模型版本。如需詳細資訊，請參閱 模型版本控制。

OrientationCorrection 是映像的估計方向。如果您使用的臉部偵測模型版本大於第 3 版，則不會傳回方向更正資訊。如需詳細資訊，請參閱 取得映像方向與週框方塊座標。

下列範例回應顯示指定 ["ALL"] 時傳回的 JSON：


```
{
    "FaceModelVersion": "3.0",
    "FaceRecords": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.06333333253860474,
                    "Left": 0.17185185849666595,
                    "Top": 0.7366666793823242,
                    "Width": 0.11061728745698929
                },
                "Confidence": 99.99999237060547,
                "ExternalImageId": "input.jpg",
                "FaceId": "578e2e1b-d0b0-493c-aa39-ba476a421a34",
                "ImageId": "9ba38e68-35b6-5509-9d2e-fcffa75d1653"
            },
            "FaceDetail": {
                "AgeRange": {
                    "High": 25,
                    "Low": 15
                },
                "Beard": {
                    "Confidence": 99.98077392578125,
                    "Value": false
                },
                "BoundingBox": {
                    "Height": 0.06333333253860474,
                    "Left": 0.17185185849666595,
                    "Top": 0.7366666793823242,
                    "Width": 0.11061728745698929
                },
```

```
                "Confidence": 99.99999237060547,
                "Emotions": [
                    {
                        "Confidence": 95.40877532958984,
                        "Type": "HAPPY"
                    },
                    {
                        "Confidence": 6.6088080406188965,
                        "Type": "CALM"
                    },
                    {
                        "Confidence": 0.7385611534118652,
                        "Type": "SAD"
                    }
                ],
                 "EyeDirection": {
                    "yaw": 16.299732,
                    "pitch": -6.407457,
                    "confidence": 99.968704
                              }
                "Eyeglasses": {
                    "Confidence": 99.96795654296875,
                    "Value": false
                },
                "EyesOpen": {
                    "Confidence": 64.0671157836914,
                    "Value": true
                },
                "Gender": {
                    "Confidence": 100,
                    "Value": "Female"
                },
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.21361233294010162,
                        "Y": 0.757106363773346
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.2518567442893982,
                        "Y": 0.7599404454231262
                    },
                    {
```

```
                    "Type": "nose",
                    "X": 0.2262365221977234,
                    "Y": 0.7711842060089111
                },
                {
                    "Type": "mouthLeft",
                    "X": 0.2050037682056427,
                    "Y": 0.7801263332366943
                },
                {
                    "Type": "mouthRight",
                    "X": 0.2430567592382431,
                    "Y": 0.7836716771125793
                },
                {
                    "Type": "leftPupil",
                    "X": 0.2161938101053238,
                    "Y": 0.756662905216217
                },
                {
                    "Type": "rightPupil",
                    "X": 0.2523181438446045,
                    "Y": 0.7603650689125061
                },
                {
                    "Type": "leftEyeBrowLeft",
                    "X": 0.20066319406032562,
                    "Y": 0.7501518130302429
                },
                {
                    "Type": "leftEyeBrowUp",
                    "X": 0.2130996286869049,
                    "Y": 0.7480520606040955
                },
                {
                    "Type": "leftEyeBrowRight",
                    "X": 0.22584207355976105,
                    "Y": 0.7504606246948242
                },
                {
                    "Type": "rightEyeBrowLeft",
                    "X": 0.24509544670581818,
                    "Y": 0.7526801824569702
                },
```

```
                    {
                        "Type": "rightEyeBrowUp",
                        "X": 0.2582615911960602,
                        "Y": 0.7516844868659973
                    },
                    {
                        "Type": "rightEyeBrowRight",
                        "X": 0.26881539821624756,
                        "Y": 0.7554477453231812
                    },
                    {
                        "Type": "leftEyeLeft",
                        "X": 0.20624476671218872,
                        "Y": 0.7568746209144592
                    },
                    {
                        "Type": "leftEyeRight",
                        "X": 0.22105035185813904,
                        "Y": 0.7582521438598633
                    },
                    {
                        "Type": "leftEyeUp",
                        "X": 0.21401576697826385,
                        "Y": 0.7553104162216187
                    },
                    {
                        "Type": "leftEyeDown",
                        "X": 0.21317370235919952,
                        "Y": 0.7584449648857117
                    },
                    {
                        "Type": "rightEyeLeft",
                        "X": 0.24393919110298157,
                        "Y": 0.7600628137588501
                    },
                    {
                        "Type": "rightEyeRight",
                        "X": 0.2598416209220886,
                        "Y": 0.7605880498886108
                    },
                    {
                        "Type": "rightEyeUp",
                        "X": 0.2519053518772125,
                        "Y": 0.7582084536552429
```

```
            },
            {
                "Type": "rightEyeDown",
                "X": 0.25177454948425293,
                "Y": 0.7612871527671814
            },
            {
                "Type": "noseLeft",
                "X": 0.2185886949300766,
                "Y": 0.774715781211853
            },
            {
                "Type": "noseRight",
                "X": 0.23328955471515656,
                "Y": 0.7759330868721008
            },
            {
                "Type": "mouthUp",
                "X": 0.22446128726005554,
                "Y": 0.7805567383766174
            },
            {
                "Type": "mouthDown",
                "X": 0.22087252140045166,
                "Y": 0.7891407608985901
            }
        ],
        "MouthOpen": {
            "Confidence": 95.87068939208984,
            "Value": false
        },
        "Mustache": {
            "Confidence": 99.9828109741211,
            "Value": false
        },
        "Pose": {
            "Pitch": -0.9409101605415344,
            "Roll": 7.233824253082275,
            "Yaw": -2.3602254390716553
        },
        "Quality": {
            "Brightness": 32.01998519897461,
            "Sharpness": 93.67259216308594
        },
```

```
                "Smile": {
                    "Confidence": 86.7142105102539,
                    "Value": true
                },
                "Sunglasses": {
                    "Confidence": 97.38925170898438,
                    "Value": false
                }
            }
        }
    ],
    "OrientationCorrection": "ROTATE_0"
    "UnindexedFaces": []
}
```

列出集合中的人臉和關聯使用者

您可以使用此ListFaces作業列示集合中的面及其關聯使用者。

如需詳細資訊，請參閱 管理集合中的人臉。

若要列出集合中的臉部 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 ListFaces 操作。

   Java

   此範例顯示集合中的人臉的清單。

   將 collectionId 的值變更為所需的集合。

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

```
package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Face;
import com.amazonaws.services.rekognition.model.ListFacesRequest;
import com.amazonaws.services.rekognition.model.ListFacesResult;
import java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;



public class ListFacesInCollection {
    public static final String collectionId = "MyCollection";

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

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ObjectMapper objectMapper = new ObjectMapper();

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

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

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

         listFacesResult =  rekognitionClient.listFaces(listFacesRequest);
         List < Face > faces = listFacesResult.getFaces();
         for (Face face: faces) {
            System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
               .writeValueAsString(face));
         }
      } while (listFacesResult != null && listFacesResult.getNextToken() !=
         null);
   }
```

```
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
// snippet-start:[rekognition.java2.list_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Face;
import software.amazon.awssdk.services.rekognition.model.ListFacesRequest;
import software.amazon.awssdk.services.rekognition.model.ListFacesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
// snippet-end:[rekognition.java2.list_faces_collection.import]

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

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId>\n\n" +
            "Where:\n" +
            "   collectionId - The name of the collection. \n\n";

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

```
        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Faces in collection " + collectionId);
        listFacesCollection(rekClient, collectionId) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.list_faces_collection.main]
    public static void listFacesCollection(RekognitionClient rekClient, String
 collectionId ) {
        try {
            ListFacesRequest facesRequest = ListFacesRequest.builder()
                .collectionId(collectionId)
                .maxResults(10)
                .build();

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

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
         }
      }
    // snippet-end:[rekognition.java2.list_faces_collection.main]
  }
```

AWS CLI

此 AWS CLI 命令會顯示 `list-faces` CLI 作業的 JSON 輸出。將 `collection-id` 的值取代為要列出的集合的名稱。將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

```
aws rekognition list-faces --collection-id "collection-id"  --profile profile-
name
```

Python

此範例顯示集合中的人臉的清單。

將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

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

import boto3

def list_faces_in_collection(collection_id):
    maxResults = 2
    faces_count = 0
    tokens = True

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')
    response = client.list_faces(CollectionId=collection_id,
                                 MaxResults=maxResults)

    print('Faces in collection ' + collection_id)

    while tokens:

        faces = response['Faces']

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

```
def main():
    collection_id = 'collection-id'
    faces_count = list_faces_in_collection(collection_id)
    print("faces count: " + str(faces_count))

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

.NET

此範例顯示集合中的人臉的清單。

將 collectionId 的值變更為所需的集合。

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

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

public class ListFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        ListFacesResponse listFacesResponse = null;
        Console.WriteLine("Faces in collection " + collectionId);

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

            ListFacesRequest listFacesRequest = new ListFacesRequest()
            {
```

```
                CollectionId = collectionId,
                MaxResults = 1,
                NextToken = paginationToken
            };

            listFacesResponse = rekognitionClient.ListFaces(listFacesRequest);
            foreach(Face face in listFacesResponse.Faces)
                Console.WriteLine(face.FaceId);
        } while (listFacesResponse != null && !
String.IsNullOrEmpty(listFacesResponse.NextToken));
    }
}
```

ListFaces 操作請求

ListFaces 的輸入值是您要為其列出人臉的集合 ID。MaxResults 是要傳回之人臉的最大值。ListFaces 還包含面部 ID 列表以過濾結果，並提供用戶 ID 以僅列出與給定用戶關聯的臉部。

```
{
    "CollectionId": "MyCollection",
    "MaxResults": 1
}
```

如果回應的人臉多於 MaxResults 請求的人臉，則會傳回一個字符，您可以在後續的 ListFaces 呼叫中使用該字符獲取下一組結果。例如：

```
{
    "CollectionId": "MyCollection",
    "NextToken": "sm+5ythT3aeEVIR4WA....",
    "MaxResults": 1
}
```

ListFaces 作業回應

ListFaces 的回應是有關存放在指定集合中的臉部中繼資料的資訊。

- FaceModelVersion— 與集合相關聯的臉部模型版本。如需詳細資訊，請參閱 模型版本控制。
- Faces：有關集合中的人臉資訊。這包括有關 BoundingBox，可信度，圖像標識符和面部 ID 的信息。如需詳細資訊，請參閱人臉。
- NextToken-用於獲取下一組結果的令牌。

```
{
    "FaceModelVersion": "6.0",
    "Faces": [
        {
            "Confidence": 99.76940155029297,
            "IndexFacesModelVersion": "6.0",
            "UserId": "demoUser2",
            "ImageId": "56a0ca74-1c83-39dd-b363-051a64168a65",
            "BoundingBox": {
                "Width": 0.03177810087800026,
                "Top": 0.36568498611450195,
                "Left": 0.3453829884529114,
                "Height": 0.056759100407361984
            },
            "FaceId": "c92265d4-5f9c-43af-a58e-12be0ce02bc3"
        },
        {
            "BoundingBox": {
                "Width": 0.03254450112581253,
                "Top": 0.6080359816551208,
                "Left": 0.5160620212554932,
                "Height": 0.06347999721765518
            },
            "IndexFacesModelVersion": "6.0",
            "FaceId": "851cb847-dccc-4fea-9309-9f4805967855",
            "Confidence": 99.94369506835938,
            "ImageId": "a8aed589-ceec-35f7-9c04-82e0b546b024"
        },
        {
            "BoundingBox": {
                "Width": 0.03094629943370819,
                "Top": 0.4218429923057556,
                "Left": 0.6513839960098267,
                "Height": 0.05266290158033371
            },
            "IndexFacesModelVersion": "6.0",
            "FaceId": "c0eb3b65-24a0-41e1-b23a-1908b1aaeac1",
            "Confidence": 99.82969665527344,
            "ImageId": "56a0ca74-1c83-39dd-b363-051a64168a65"
        }
    ]
}
```

刪除集合中的人臉

您可以使用此DeleteFaces作業從集合中刪除面。如需詳細資訊，請參閱 管理集合中的人臉。

刪除集合中的臉部

1. 如果您尚未執行：
   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。
   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用下列範例來呼叫 DeleteFaces 操作。

   Java

   此範例刪除集合中的單一人臉。

   將 collectionId 的值變更為包含要刪除的人臉的集合。將 faces 的值變更為要刪除的人臉 ID。如果要刪除多個人臉相符，可新增人臉 ID 至 faces 陣列。

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

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteFacesRequest;
import com.amazonaws.services.rekognition.model.DeleteFacesResult;

import java.util.List;


public class DeleteFacesFromCollection {
   public static final String collectionId = "MyCollection";
   public static final String faces[] = {"xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"};

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

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


      DeleteFacesRequest deleteFacesRequest = new DeleteFacesRequest()
              .withCollectionId(collectionId)
              .withFaceIds(faces);

      DeleteFacesResult
 deleteFacesResult=rekognitionClient.deleteFaces(deleteFacesRequest);


      List < String > faceRecords = deleteFacesResult.getDeletedFaces();
      System.out.println(Integer.toString(faceRecords.size()) + " face(s)
 deleted:");
      for (String face: faceRecords) {
         System.out.println("FaceID: " + face);
      }
   }
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteFacesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
// snippet-end:[rekognition.java2.delete_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFacesFromCollection {
    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> <faceId> \n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection from which faces are
 deleted. \n\n" +
            "   faceId - The id of the face to delete. \n\n";

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

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteFacesCollection(rekClient, collectionId, faceId);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.delete_faces_collection.main]
    public static void deleteFacesCollection(RekognitionClient rekClient,
                                             String collectionId,
                                             String faceId) {

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

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

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.delete_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令會顯示 delete-faces CLI 作業的 JSON 輸出。將 collection-id 的值取代為包含要刪除的臉部的集合的名稱。將 face-ids 的值取代為要刪除的臉部 ID 陣列。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
aws rekognition delete-faces --collection-id "collection-id" --face-ids "faceid"
 --profile profile-name
```

Python

此範例刪除集合中的單一人臉。

將 collectionId 的值變更為包含要刪除的人臉的集合。將 faces 的值變更為要刪除的人臉 ID。如果要刪除多個人臉相符，可新增人臉 ID 至 faces 陣列。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3

def delete_faces_from_collection(collection_id, faces):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')
    response = client.delete_faces(CollectionId=collection_id,
                                   FaceIds=faces)
```

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


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

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

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

.NET

此範例刪除集合中的單一人臉。

將 collectionId 的值變更為包含要刪除的人臉的集合。將 faces 的值變更為要刪除的人臉 ID。如果要刪除多個臉部相符，可新增臉部 ID 至 faces 清單。

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

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

public class DeleteFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        List<String> faces = new List<String>() { "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" };

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();
```

```
        DeleteFacesRequest deleteFacesRequest = new DeleteFacesRequest()
        {
            CollectionId = collectionId,
            FaceIds = faces
        };

        DeleteFacesResponse deleteFacesResponse =
 rekognitionClient.DeleteFaces(deleteFacesRequest);
        foreach (String face in deleteFacesResponse.DeletedFaces)
            Console.WriteLine("FaceID: " + face);
    }
}
```

DeleteFaces 操作請求

DeleteFaces 的輸入值是包含人臉的集合 ID，以及要刪除的人臉的人臉 ID 陣列。

```
{
    "CollectionId": "MyCollection",
    "FaceIds": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

DeleteFaces 作業回應

DeleteFaces 回應包含已刪除臉部的臉部 ID 陣列。

```
{
    "DeletedFaces": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

如果輸入中提供的臉部 ID 目前與使用者相關聯，則會以正當理由 UnsuccessfulFaceDeletions 的方式傳回它們。

```
{
   "DeletedFaces": [
```

```
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ],
     "UnsuccessfulFaceDeletions" : [
        {
           "FaceId" : "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
           "UserId" : "demoUser1",
           "Reason" : ["ASSOCIATED_TO_AN_EXISTING_USER"]
        }
     ]
}
```

建立使用者

您可以使用此CreateUser作業，使用您提供的唯一使用者 ID 在集合中建立新使用者。然後，您可以將多個人臉與新建立的使用者相關聯。

建立使用者 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 權限的 IAM 使用者賬戶。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 CreateUser 操作。

   Java

   此 Java 程式碼範例建立一個使用者。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CreateUserRequest;
import com.amazonaws.services.rekognition.model.CreateUserResult;


public class CreateUser {

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

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

        //Replace collectionId and userId with the name of the user that you
 want to create in that target collection.

        String collectionId = "MyCollection";
        String userId = "demoUser";
        System.out.println("Creating new user: " +
                userId);

        CreateUserRequest request = new CreateUserRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        rekognitionClient.createUser(request);
    }

}
```

AWS CLI

此 AWS CLI 命令會使用 `create-user` CLI 作業建立使用者。

```
aws rekognition create-user --user-id user-id --collection-id collection-name --
region region-name
--client-request-token request-token
```

Python

此 Python 程式碼範例建立使用者。

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

import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')
```

```
def create_user(collection_id, user_id):
    """
    Creates a new User within a collection specified by CollectionId.
    Takes UserId as a parameter, which is a user provided ID which
    should be unique within the collection.

    :param collection_id: The ID of the collection where the indexed faces will
 be stored at.
    :param user_id: ID for the UserID to be created. This ID needs to be unique
 within the collection.

    :return: The indexFaces response
    """
    try:
        logger.info(f'Creating user: {collection_id}, {user_id}')
        client.create_user(
            CollectionId=collection_id,
            UserId=user_id
        )
    except ClientError:
        logger.exception(f'Failed to create user with given user id:
 {user_id}')
        raise

def main():
    collection_id = "collection-id"
    user_id = "user-id"
    create_user(collection_id, user_id)

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

刪除使用者

您可以使用該DeleteUser操作根據提供的 UserID 從集合中刪除用戶。請注意，在刪除指定的 UserID 之前，與 UserID 相關聯的任何人臉都會與 UserID 斷開關聯。

刪除使用者 (SDK)

1. 如果您尚未執行：

a. 建立或更新具有 AmazonRekognitionFullAccess 權限的 IAM 使用者賬戶。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 DeleteUser 操作。

Java

此 Java 程式碼範例會刪除使用者。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteUserRequest;
import com.amazonaws.services.rekognition.model.DeleteUserResult;


public class DeleteUser {

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


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        //Replace collectionId and userId with the name of the user that you
 want to delete from that target collection.

        String collectionId = "MyCollection";
        String userId = "demoUser";
        System.out.println("Deleting existing user: " +
                userId);

        DeleteUserRequest request = new DeleteUserRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        rekognitionClient.deleteUser(request);
    }

}
```

AWS CLI

此 AWS CLI 命令會使用 `create-user` CLI 作業刪除使用者。

```
aws rekognition delete-user --collection-id MyCollection
--user-id user-id --collection-id collection-name --region region-name
```

Python

此 Python 程式碼範例會刪除使用者。

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

import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def delete_user(collection_id, user_id):
        """
        Delete the user from the given collection

        :param collection_id: The ID of the collection where user is stored.
        :param user_id: The ID of the user in the collection to delete.
        """
        logger.info(f'Deleting user: {collection_id}, {user_id}')
        try:
            client.delete_user(
                CollectionId=collection_id,
                UserId=user_id
            )
        except ClientError:
            logger.exception(f'Failed to delete user with given user id:
 {user_id}')
            raise

def main():
```

```
    collection_id = "collection-id"
    user_id = "user-id"
    delete_user(collection_id, user_id)

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

將人臉與使用者產生關聯

您可以使用此AssociateFaces作業將多個個別面與單一使用者相關聯。若要將人臉與使用者建立關聯，必須先建立集合和使用者。請注意，人臉向量必須位於使用者向量所在的同一個集合中。

若要關聯人臉 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 AssociateFaces 操作。

   Java

   此 Java 程式碼範例將一個人臉與使用者相關聯。

```
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AssociateFacesRequest;
import com.amazonaws.services.rekognition.model.AssociateFacesResult;


public class AssociateFaces {

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

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

        /* Replace the below configurations to allow you successfully run the
example

           @collectionId: The collection where user and faces are stored
           @userId: The user which faces will get associated to
           @faceIds: The list of face IDs that will get associated to the given
user
           @userMatchThreshold: Minimum User match confidence required for the
face to
                                be associated with a User that has at least one
faceID already associated
         */

        String collectionId = "MyCollection";
        String userId = "demoUser";
        String faceId1 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
        String faceId2 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
        List<String> faceIds = Arrays.asList(faceid1,faceid2);

        float userMatchThreshold = 0f;
        System.out.println("Associating faces to the existing user: " +
                userId);

        AssociateFacesRequest request = new AssociateFacesRequest()
                .withCollectionId(collectionId)
                .withUserId(userId)
                .withFaceIds(faceIds)
                .withUserMatchThreshold(userMatchThreshold);

        AssociateFacesResult result = rekognitionClient.associateFaces(request);

        System.out.println("Successful face associations: " +
result.getAssociatedFaces().size());
        System.out.println("Unsuccessful face associations: " +
result.getUnsuccessfulFaceAssociations().size());
    }

}
```

AWS CLI

此 AWS CLI 指令會使用 associate-faces CLI 作業將臉孔與使用者產生關聯。

```
aws rekognition associate-faces --user-id user-id --face-ids face-id-1 face-id-2
--collection-id collection-name
--region region-name
```

Python

這個 Python 程式碼範例將一個人臉與使用者相關聯。

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def associate_faces(collection_id, user_id, face_ids):
    """
    Associate stored faces within collection to the given user

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to associate faces to
    :param face_ids: The list of face IDs to be associated to the given user

    :return: response of AssociateFaces API
    """
    logger.info(f'Associating faces to user: {user_id}, {face_ids}')
    try:
        response = client.associate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- associated {len(response["AssociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to associate faces to the given user")
        raise
```

```
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    associate_faces(collection_id, user_id, face_ids)

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

AssociateFaces 作業回應

AssociateFaces 的回應包括 UserStatus，也就是解除關聯要求的狀態，以及要關聯的 FaceIds 清單。也會傳回 UnsuccessfulFaceAssociations 的清單。將請求提交給 AssociateFaces 之後，操作可能需要一分鐘左右的時間才能完成。

基於這個原因 UserStatus ，返回，它可以具有以下值：

- 已建立：表示「使用者」已成功建立，且目前沒有與其相關聯的任何人臉。在進行任何成功的 " 呼叫之前，'使用者AssociateFaces' 將處於此狀態。
- 更新：表示「使用者」正在更新以反映新關聯/取消關聯的人臉，並在幾秒鐘內變為作用中狀態。此狀態下的搜尋結果可能包含「使用者」，客戶可以選擇從傳回的結果中忽略這些結果。
- 作用中：表示「使用者」已更新，以反映所有關聯/已取消關聯的人臉，且處於可搜尋狀態。

```
{
    "UnsuccessfulFaceAssociations": [
        {
            "Reasons": [
                "LOW_MATCH_CONFIDENCE"
            ],
            "FaceId": "f5817d37-94f6-0000-bfee-1a2b3c4d5e6f",
            "Confidence": 0.9375374913215637
        },
        {
            "Reasons": [
                "ASSOCIATED_TO_A_DIFFERENT_IDENTITY"
            ],
```

```
            "FaceId": "851cb847-dccc-1111-bfee-1a2b3c4d5e6f",
            "UserId": "demoUser2"
        }
    ],
    "UserStatus": "UPDATING",
    "AssociatedFaces": [
        {
            "FaceId": "35ebbb41-7f67-2222-bfee-1a2b3c4d5e6f"
        }
    ]
}
```

取消使用者人臉的關聯

您可以使用此DisassociateFaces作業移除使用者 ID 和臉部識別碼之間的關聯。

取消人臉的關聯 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 DisassociateFaces 操作。

   Java

   此 Java 範例會移除 FaceID 和 UserID 與 DisassociateFaces 操作之間的關聯。

```
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DisassociateFacesRequest;
import com.amazonaws.services.rekognition.model.DisassociateFacesResult;


public class DisassociateFaces {
```

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


       AmazonRekognition rekognitionClient =
AmazonRekognitionClientBuilder.defaultClient();

       /* Replace the below configurations to allow you successfully run the
example

          @collectionId: The collection where user and faces are stored
          @userId: The user which faces will get disassociated from
          @faceIds: The list of face IDs that will get disassociated from the
given user
        */

       String collectionId = "MyCollection";
       String userId = "demoUser";
       String faceId1 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
       String faceId2 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
       List<String> faceIds = Arrays.asList(faceid1,faceid2);

       System.out.println("Disassociating faces from existing user: " +
               userId);

       DisassociateFacesRequest request = new DisassociateFacesRequest()
               .withCollectionId(collectionId)
               .withUserId(userId)
               .withFaceIds(faceIds)

       DisassociateFacesResult result =
rekognitionClient.disassociateFaces(request);

       System.out.println("Successful face disassociations: " +
result.getDisassociatedFaces().size());
       System.out.println("Unsuccessful face disassociations: " +
result.getUnsuccessfulFaceDisassociations().size());
   }

}
```

AWS CLI

此 AWS CLI 命令會移除 FaceID 和 UserID 與作業之間的關聯。DisassociateFaces

```
aws rekognition disassociate-faces --face-ids list-of-face-ids
--user-id user-id --collection-id collection-name --region region-name
```

Python

下列範例會移除 FaceID 和 UserID 與 DisassociateFaces 操作之間的關聯。

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def disassociate_faces(collection_id, user_id, face_ids):
    """
    Disassociate stored faces within collection to the given user

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to disassociate faces from
    :param face_ids: The list of face IDs to be disassociated from the given
 user

    :return: response of AssociateFaces API
    """
    logger.info(f'Disssociating faces from user: {user_id}, {face_ids}')
    try:
        response = client.disassociate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- disassociated {len(response["DisassociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to disassociate faces from the given user")
        raise
```

```
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    disassociate_faces(collection_id, user_id, face_ids)

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

DisassociateFaces 作業回應

DisassociateFaces 的回應包括 UserStatus，也就是解除關聯要求的狀態，以及要取消關聯 FaceIds 的清單。也會傳回 UnsuccessfulFaceDisassociations 的清單。將請求提交給之後 DisassociateFaces，操作可能需要一分鐘左右的時間才能完成。基於這個原因 UserStatus ，返回，它可以具有以下值：

- 已建立：表示「使用者」已成功建立，且目前沒有與其相關聯的任何人臉。在進行任何成功的 " 呼叫之前，'使用者AssociateFaces' 將處於此狀態。
- 更新：表示「使用者」正在更新以反映新關聯/取消關聯的人臉，並在幾秒鐘內變為作用中狀態。此狀態下的搜尋結果可能包含「使用者」，客戶可以選擇從傳回的結果中忽略這些結果。
- 作用中：表示「使用者」已更新，以反映所有關聯/已取消關聯的人臉，且處於可搜尋狀態。

```
{
    "UserStatus": "UPDATING",
    "DisassociatedFaces": [
        {
            "FaceId": "c92265d4-5f9c-43af-a58e-12be0ce02bc3"
        }
    ],
    "UnsuccessfulFaceDisassociations": [
        {
            "Reasons": [
                "ASSOCIATED_TO_A_DIFFERENT_IDENTITY"
            ],
            "FaceId": "f5817d37-94f6-4335-bfee-6cf79a3d806e",
```

```
            "UserId": "demoUser1"
        }
    ]
}
```

在集合中列出人臉

您可以使用ListUsers作業來列出 UserIds 和 UserStatus。若要查看與 UserID 相關聯的 FaceiID，請使用此作業。ListFaces

若要列出使用者 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 ListUsers 操作。

   Java

   此 Java 範例列出了使用 ListUsers 操作的集合中的使用者。

```
import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.ListUsersRequest;
import com.amazonaws.services.rekognition.model.ListUsersResult;
import com.amazonaws.services.rekognition.model.User;

public class ListUsers {

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


        AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();


        System.out.println("Listing users");
```

```
        int limit = 10;
        ListUsersResult listUsersResult = null;
        String paginationToken = null;
        do {
            if (listUsersResult != null) {
                paginationToken = listUsersResult.getNextToken();
            }
            ListUsersRequest request = new ListUsersRequest()
                    .withCollectionId(collectionId)
                    .withMaxResults(limit)
                    .withNextToken(paginationToken);
            listUsersResult = amazonRekognition.listUsers(request);

            List<User> users = listUsersResult.getUsers();
            for (User currentUser: users) {
                System.out.println(currentUser.getUserId() + " : " +
 currentUser.getUserStatus());
            }
        } while (listUsersResult.getNextToken() != null);
    }
}
```

AWS CLI

此 AWS CLI 命令列出了具有ListUsers操作的集合中的用戶。

```
aws rekognition list-users --collection-id collection-id --max-results number-
of-max-results
```

Python

下列範例會列出具有 ListUsers 操作的集合中的使用者。

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

import boto3
from botocore.exceptions import ClientError
import logging
from pprint import pprint

logger = logging.getLogger(__name__)
```

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


def list_users(collection_id):
    """
    List all users from the given collection

    :param collection_id: The ID of the collection where user is stored.

    :return: response that contains list of Users found within given collection
    """
    logger.info(f'Listing the users in collection: {collection_id}')
    try:
        response = client.list_users(
            CollectionId=collection_id
        )
        pprint(response["Users"])
    except ClientError:
        logger.exception(f'Failed to list all user from given collection:
 {collection_id}')
        raise
    else:
        return response

def main():
    collection_id = "collection-id"
    list_users(collection_id)

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

ListUsers 作業回應

要求的回應 ListUsers 包含集合Users中的清單，以UsedId及使用者UserStatus的和。

```
{
    "NextToken": "B1asJT3bAb/ttuGgPFV8BZoBZyGQzlUHXbuTNLh48a6enU7kXKw43hpOwizW7LOk/
Gk7Em09lznoq6+FcDCcSq2olrn7A98BLkt5keu+ZRVRUTyrXtT6J7Hmp
+ieQ2an6Zu0qzPfcdPeaJ9eAxG2d0WNrzJgi5hvmjoiSTTfKX3MQz1sduWQkvAAs4hZfhZoKFahFlqWofshCXa/
FHAAY3PL1PjxXbkNeSSMq8V7i1MlKCdrPVykCv9MokpPt7jtNvKPEZGUhxgBTFMxNWLEcFnzAiCWDg91dFy/
```

```
LalshPjXA9UVc5Gx9vIJNQ/
eO3cQRghAkCT3FOAiXsLAnA015ODTomZpWWVpqB21wKpI3LYmfAVFrDPGzpbTVlRmLsJm41bkmnBBBw9+DHz1Jn7zW
+qc5Fs3yaHu0f51Xg==",
    "Users": [
        {
            "UserId": "demoUser4",
            "UserStatus": "CREATED"
        },
        {
            "UserId": "demoUser2",
            "UserStatus": "CREATED"
        }
    ]
}
```

搜尋具有人臉 ID 的人臉

您可以使用此SearchFaces作業搜尋集合中符合所提供影像中最大臉孔的使用者。

偵測到臉孔並將其新增至集合時，會在IndexFaces作業回應中傳回臉部 ID。如需詳細資訊，請參閱 管理集合中的人臉。

若要使用人臉 ID 來搜尋集合中的人臉 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 SearchFaces 操作。

   Java

   此範例顯示有關與由其 ID 識別的人臉符合的人臉資訊。

   將 collectionID 的值變更為包含所需人臉的集合。將 faceId 的值變更為要尋找的人臉識別符。

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

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.SearchFacesRequest;
import com.amazonaws.services.rekognition.model.SearchFacesResult;
import java.util.List;


  public class SearchFaceMatchingIdCollection {
      public static final String collectionId = "MyCollection";
      public static final String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx";

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

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

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

       SearchFacesResult searchFacesByIdResult =
               rekognitionClient.searchFaces(searchFacesRequest);

       System.out.println("Face matching faceId " + faceId);
      List < FaceMatch > faceImageMatches =
 searchFacesByIdResult.getFaceMatches();
      for (FaceMatch face: faceImageMatches) {
         System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
                 .writeValueAsString(face));
```

```
            System.out.println();
        }
    }

}
```

執行程式碼範例。顯示有關人臉符合的資訊。

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
// snippet-start:[rekognition.java2.match_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.SearchFacesRequest;
import software.amazon.awssdk.services.rekognition.model.SearchFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
// snippet-end:[rekognition.java2.match_faces_collection.import]

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

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> <sourceImage>\n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection.  \n" +
```

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

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

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Searching for a face in a collections");
        searchFacebyId(rekClient, collectionId, faceId ) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.match_faces_collection.main]
    public static void searchFacebyId(RekognitionClient rekClient,String
 collectionId, String faceId) {

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

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

```
        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.match_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令會顯示 search-faces CLI 作業的 JSON 輸出。以您想要搜尋的臉部識別碼來取代face-id 的值，並以您想要搜尋的集合來取代 collection-id 的值。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
aws rekognition search-faces --face-id face-id --collection-id "collection-id"
 --profile profile-name
```

Python

此範例顯示有關與由其 ID 識別的人臉符合的人臉資訊。

將 collectionID 的值變更為包含所需人臉的集合。將 faceId 的值變更為要尋找的人臉識別符。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3

def search_face_in_collection(face_id, collection_id):
    threshold = 90
    max_faces = 2

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

    response = client.search_faces(CollectionId=collection_id,
                                   FaceId=face_id,
                                   FaceMatchThreshold=threshold,
                                   MaxFaces=max_faces)
```

```
    face_matches = response['FaceMatches']
    print('Matching faces')
    for match in face_matches:
        print('FaceId:' + match['Face']['FaceId'])
        print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")

    return len(face_matches)

def main():
    face_id = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
    collection_id = 'collection-id'

    faces = []
    faces.append(face_id)

    faces_count = search_face_in_collection(face_id, collection_id)
    print("faces found: " + str(faces_count))

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

.NET

此範例顯示有關與由其 ID 識別的人臉符合的人臉資訊。

將 collectionID 的值變更為包含所需人臉的集合。將 faceId 的值變更為要尋找的臉部識別符。

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

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

public class SearchFacesMatchingId
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
```

```
        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        // Search collection for faces matching the face id.

        SearchFacesRequest searchFacesRequest = new SearchFacesRequest()
        {
            CollectionId = collectionId,
            FaceId = faceId,
            FaceMatchThreshold = 70F,
            MaxFaces = 2
        };

        SearchFacesResponse searchFacesResponse =
 rekognitionClient.SearchFaces(searchFacesRequest);

        Console.WriteLine("Face matching faceId " + faceId);

        Console.WriteLine("Matche(s): ");
        foreach (FaceMatch face in searchFacesResponse.FaceMatches)
            Console.WriteLine("FaceId: " + face.Face.FaceId + ", Similarity: " +
 face.Similarity);
    }
}
```

執行程式碼範例。顯示有關人臉符合的資訊。

SearchFaces 操作請求

SearchFaces 將根據人臉 ID (每個儲存在人臉集合中的人臉都有人臉 ID) 在指定的人臉集合中搜尋相似的人臉。回應不包含您要搜尋的臉部。它只包含類似的臉部。在預設情況下，SearchFaces 傳回在使用演算法偵測後相似度超過 80% 的人臉。相似度代表偵測到的人臉與輸入人臉間符合的程度。或者，可以使用 FaceMatchThreshold 來指定不同的值。

```
{
    "CollectionId": "MyCollection",
    "FaceId": "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFaces 作業回應

此操作會傳回一系列找到的人臉配對以及您提供的人臉 ID 來做為輸入檔。

```
{
    "SearchedFaceId": "7ecf8c19-5274-5917-9c91-1db9ae0449e2",
    "FaceMatches": [ list of face matches found ]
}
```

對於每個找到的臉部配對，回應將包含相似度以及臉部中繼資料，如下方範例回應所示：

```
{
   ...
    "FaceMatches": [
        {
            "Similarity": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.6154,
                    "Top": 0.2442,
                    "Left": 0.1765,
                    "Height": 0.4692
                },
                "FaceId": "84de1c86-5059-53f2-a432-34ebb704615d",
                "Confidence": 99.9997,
                "ImageId": "d38ebf91-1a11-58fc-ba42-f978b3f32f60"
            }
        },
        {
            "Similarity": 84.6859,
            "Face": {
                "BoundingBox": {
                    "Width": 0.2044,
                    "Top": 0.2254,
                    "Left": 0.4622,
                    "Height": 0.3119
                },
                "FaceId": "6fc892c7-5739-50da-a0d7-80cc92c0ba54",
                "Confidence": 99.9981,
                "ImageId": "5d913eaf-cf7f-5e09-8c8f-cb1bdea8e6aa"
            }
        }
    ]
```

```
}
```

使用映像來搜尋人臉

您可以使用此SearchFacesByImage作業在集合中搜尋符合所提供影像中最大面的面的面。

如需詳細資訊，請參閱 在集合中搜尋人臉與使用者。

若要使用映像來搜尋集合中的人臉 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 將 (含有一或多個人臉) 的映像上傳至您的 S3 儲存貯體。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例來呼叫 SearchFacesByImage 操作。

   Java

   此範例顯示有關與映像中最大人臉符合的人臉資訊。程式碼範例指定 FaceMatchThreshold 和 MaxFaces 參數以限制回應中傳回的結果。

   在下列範例中，變更下列內容：將 collectionId 的值變更為要搜尋的集合，將 bucket 的值變更為包含輸入映像的儲存貯體，然後將 photo 的值變更為輸入映像。

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

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
```

```
import com.amazonaws.services.rekognition.model.SearchFacesByImageRequest;
import com.amazonaws.services.rekognition.model.SearchFacesByImageResult;
import java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;


public class SearchFaceMatchingImageCollection {
    public static final String collectionId = "MyCollection";
    public static final String bucket = "bucket";
    public static final String photo = "input.jpg";

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

       AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ObjectMapper objectMapper = new ObjectMapper();

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

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

       SearchFacesByImageResult searchFacesByImageResult =
               rekognitionClient.searchFacesByImage(searchFacesByImageRequest);

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

```
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
// snippet-start:[rekognition.java2.search_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
// snippet-end:[rekognition.java2.search_faces_collection.import]

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

    public static void main(String[] args) {

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

```
            "   <collectionId> <sourceImage>\n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection.  \n" +
            "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

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

        String collectionId = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Searching for a face in a collections");
        searchFaceInCollection(rekClient, collectionId, sourceImage ) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.search_faces_collection.main]
    public static void searchFaceInCollection(RekognitionClient rekClient,String
 collectionId, String sourceImage) {

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

            SearchFacesByImageRequest facesByImageRequest =
 SearchFacesByImageRequest.builder()
                .image(souImage)
                .maxFaces(10)
                .faceMatchThreshold(70F)
                .collectionId(collectionId)
                .build();
```

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

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.search_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令會顯示 `search-faces-by-image` CLI 作業的 JSON 輸出。以您在步驟 2 中所使用的 S3 儲存貯體來取代 `Bucket` 的值。以您在步驟 2 中所使用的映像檔案名稱來取代 `Name` 的值。以您要搜尋的集合名稱取代 `collection-id` 的值。將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

```
aws rekognition search-faces-by-image --image '{"S3Object":{"Bucket":"bucket-
name","Name":"image-name"}}' \
--collection-id "collection-id" --profile profile-name
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
aws rekognition search-faces-by-image --image "{\"S3Object\":{\"Bucket\":
\"bucket-name\",\"Name\":\"image-name\"}}" \
--collection-id "collection-id" --profile profile-name
```

Python

此範例顯示有關與映像中最大人臉符合的人臉資訊。程式碼範例指定 FaceMatchThreshold 和 MaxFaces 參數以限制回應中傳回的結果。

在下列範例中，變更下列內容：將 collectionId 的值變更為要搜尋的集合，並將 bucket 和 photo 的值取代為步驟 2 中使用的 Amazon S3 儲存貯體和映像的名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

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

import boto3

if __name__ == "__main__":

    bucket='bucket'
    collectionId='MyCollection'
    fileName='input.jpg'
    threshold = 70
    maxFaces=2

    client=boto3.client('rekognition')


    response=client.search_faces_by_image(CollectionId=collectionId,
                                Image={'S3Object':
{'Bucket':bucket,'Name':fileName}},
                                FaceMatchThreshold=threshold,
                                MaxFaces=maxFaces)


    faceMatches=response['FaceMatches']
    print ('Matching faces')
    for match in faceMatches:
            print ('FaceId:' + match['Face']['FaceId'])
            print ('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
            print
```

.NET

此範例顯示有關與映像中最大人臉符合的人臉資訊。程式碼範例指定 FaceMatchThreshold 和 MaxFaces 參數以限制回應中傳回的結果。

在下列範例中，變更下列內容：將 collectionId 的值變更為要搜尋的集合，並將 bucket 和 photo 的值取代為步驟 2 中使用的 Amazon S3 儲存貯體和映像的名稱。

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        Image image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo
            }
        };

        SearchFacesByImageRequest searchFacesByImageRequest = new
 SearchFacesByImageRequest()
        {
            CollectionId = collectionId,
            Image = image,
```

```
            FaceMatchThreshold = 70F,
            MaxFaces = 2
        };

        SearchFacesByImageResponse searchFacesByImageResponse =
 rekognitionClient.SearchFacesByImage(searchFacesByImageRequest);

        Console.WriteLine("Faces matching largest face in image from " + photo);
        foreach (FaceMatch face in searchFacesByImageResponse.FaceMatches)
            Console.WriteLine("FaceId: " + face.Face.FaceId + ", Similarity: " +
 face.Similarity);
    }
}
```

SearchFacesByImage 操作請求

SearchFacesImageByImage 的輸入參數是要搜尋的集合和來源映像位置。在此範例中，來源映像儲存於 Amazon S3 儲存貯體 (S3Object)。另外指定要傳回的最大臉數 (Maxfaces) 以及必須與要傳回的人臉 (FaceMatchThreshold) 相符的最小可信度。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFacesByImage 作業回應

指定輸入映像 (.jpeg 或 .png) 時，操作首先將偵側輸入映像中的人臉，接著搜尋指定的人臉集合以尋找相似人臉。

> **Note**
>
> 如果服務在輸入映像中偵測到多個人臉，將使用偵測到的最大人臉來搜尋人臉集合。

此操作會傳回找到的相符人臉阵列以及有關輸入人臉的資訊。這包括邊框等資訊，以及表示邊框包含人臉的可信度水平的可信度值。

在預設情況下，SearchFacesByImage 傳回在使用演算法偵測後相似度超過 80% 的人臉。相似度代表偵測到的人臉與輸入人臉間符合的程度。或者，可以使用 FaceMatchThreshold 來指定不同的值。對於每個找到的臉部配對，回應將包含相似度以及臉部中繼資料，如下方範例回應所示：

```
{
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.06333330273628235,
                    "Left": 0.1718519926071167,
                    "Top": 0.7366669774055481,
                    "Width": 0.11061699688434601
                },
                "Confidence": 100,
                "ExternalImageId": "input.jpg",
                "FaceId": "578e2e1b-d0b0-493c-aa39-ba476a421a34",
                "ImageId": "9ba38e68-35b6-5509-9d2e-fcffa75d1653"
            },
            "Similarity": 99.9764175415039
        }
    ],
    "FaceModelVersion": "3.0",
    "SearchedFaceBoundingBox": {
        "Height": 0.06333333253860474,
        "Left": 0.17185185849666595,
        "Top": 0.7366666793823242,
        "Width": 0.11061728745698929
    },
    "SearchedFaceConfidence": 99.99999237060547
}
```

搜尋使用者 (人臉識別碼/使用者 ID)

您可以使用此SearchUsers作業來搜尋指定集合中與提供的臉部 ID 或使用者 ID 相符的使用者。作業會依據要求的最高相似度分數列出傳回的UserIds排名 UserMatchThreshold。會在 CreateUsers 作業中建立使用者 ID。如需詳細資訊，請參閱 管理集合中的使用者。

若要搜尋使用者 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 SearchUsers 操作。

   Java

   此 Java 範例使用 SearchUsers 操作搜尋集合中的使用者。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.UserMatch;
import com.amazonaws.services.rekognition.model.SearchUsersRequest;
import com.amazonaws.services.rekognition.model.SearchUsersResult;
import com.amazonaws.services.rekognition.model.UserMatch;

public class SearchUsers {
    //Replace collectionId and faceId with the values you want to use.

    public static final String collectionId = "MyCollection";
    public static final String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

    public static final String userd = 'demo-user';

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        // Search collection for faces matching the user id.
        SearchUsersRequest request = new SearchUsersRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        SearchUsersResult result =
                rekognitionClient.searchUsers(request);
```

```
        System.out.println("Printing first search result with matched user and
similarity score");
        for (UserMatch match: result.getUserMatches()) {
            System.out.println(match.getUser().getUserId() + " with similarity
score " + match.getSimilarity());
        }

        // Search collection for faces matching the face id.
        SearchUsersRequest request1 = new SearchUsersRequest()
                .withCollectionId(collectionId)
                .withFaceId(faceId);

        SearchUsersResult result1 =
                rekognitionClient.searchUsers(request1);

        System.out.println("Printing second search result with matched user and
similarity score");
        for (UserMatch match: result1.getUserMatches()) {
            System.out.println(match.getUser().getUserId() + " with similarity
score " + match.getSimilarity());
        }
    }
```

AWS CLI

此 AWS CLI 命令搜索與SearchUsers操作集合中的用戶。

```
aws rekognition search-users --face-id face-id --collection-id collection-id --
region region-name
```

Python

下列範例會使用 SearchUsers 操作搜尋集合中的使用者。

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

import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
```

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

def search_users_by_face_id(collection_id, face_id):
    """
    SearchUsers operation with face ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param face_id: The ID of the face in the collection to search for.

    :return: response of SearchUsers API
    """
    logger.info(f'Searching for users using a face-id: {face_id}')
    try:
        response = client.search_users(
            CollectionId=collection_id,
            FaceId=face_id
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsers with given face id:
 {face_id}')
        raise
    else:
        print(response)
        return response

def search_users_by_user_id(collection_id, user_id):
    """
    SearchUsers operation with user ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user in the collection to search for.

    :return: response of SearchUsers API
    """
    logger.info(f'Searching for users using a user-id: {user_id}')
    try:
        response = client.search_users(
            CollectionId=collection_id,
```

```
            UserId=user_id
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsers with given face id:
 {user_id}')
        raise
    else:
        print(response)
        return response

def main():
    collection_id = "collection-id"
    user_id = "user-id"
    face_id = "face_id"
    search_users_by_face_id(collection_id, face_id)
    search_users_by_user_id(collection_id, user_id)

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

SearchUsers 操作請求

給定一個 FaceID 或用 UserID， SearchUsers 搜索指定的集合 ID 用戶匹配。依預設，會 SearchUsers 傳回相似度分數大於 80% 的使用者 ID。相似性表明 UserID 與提供的 FaceID 或 UserID 匹配的程度。如果傳回多個 UserID，則會依相似度分數最高到最低的順序列出這些 UserID。或者，您可以使 UserMatchThreshold 用指定不同的值。如需詳細資訊，請參閱 管理集合中的使用者。

以下是使用的 SearchUsers 請求示例`UserId`：

```
{
   "CollectionId": "MyCollection",
   "UserId": "demoUser1",
   "MaxUsers": 2,
   "UserMatchThreshold": 99
}
```

以下是使用的 SearchUsers 請求示例FaceId：

```
{
    "CollectionId": "MyCollection",
    "FaceId": "bff43c40-cfa7-4b94-bed8-8a08b2205107",
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsers 作業回應

如果使用搜尋FaceId，則的回應會 SearchUsers 包含的SearchedFace，以及每位使用者UserStatus的UserMatchesUserId和清單。FaceId

```
{
    "SearchedFace": {
        "FaceId": "bff43c40-cfa7-4b94-bed8-8a08b2205107"
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser1",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 100.0
        },
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6"
}
```

如果使用搜尋UserId，則除了其他回應元素之外SearchedUser，的回應還 SearchUsers 包括的。UserId

```
{
    "SearchedUser": {
        "UserId": "demoUser1"
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6"
}
```

搜尋使用者 (映像)

針對與所提供映像中偵測到的最大人臉相符的集合使用者，SearchUsersByImage 會搜尋指定的 CollectionID。依預設，會 SearchUsersByImage 傳回相似度分數大於 80% 的使用者 ID。相似性表示 UserID 與提供的映像中偵測到的最大人臉相符的程度。如果傳回多個 UserID，則會依相似度分數最高到最低的順序列出這些 UserID。或者，您可以使 UserMatchThreshold 用指定不同的值。如需詳細資訊，請參閱管理集合中的使用者。

若要依映像搜尋使用者 (SDK)

1. 如果您尚未執行：
   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。
   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用下列範例來呼叫 SearchUsersByImage 操作。

Java

此 Java 範例搜尋基於輸入映像集合中的使用者，使用 SearchUsersByImage 操作。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.SearchUsersByImageRequest;
import com.amazonaws.services.rekognition.model.SearchUsersByImageResult;
import com.amazonaws.services.rekognition.model.UserMatch;


public class SearchUsersByImage {
    //Replace bucket, collectionId and photo with your values.
    public static final String collectionId = "MyCollection";
    public static final String s3Bucket = "bucket";
    public static final String s3PhotoFileKey = "input.jpg";

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


        // Get an image object from S3 bucket.
        Image image = new Image()
                .withS3Object(new S3Object()
                        .withBucket(s3Bucket)
                        .withName(s3PhotoFileKey));

        // Search collection for users similar to the largest face in the image.
        SearchUsersByImageRequest request = new SearchUsersByImageRequest()
                .withCollectionId(collectionId)
                .withImage(image)
                .withUserMatchThreshold(70F)
                .withMaxUsers(2);

        SearchUsersByImageResult result =
                rekognitionClient.searchUsersByImage(request);

        System.out.println("Printing search result with matched user and
 similarity score");
```

```
        for (UserMatch match: result.getUserMatches()) {
            System.out.println(match.getUser().getUserId() + " with similarity
 score " + match.getSimilarity());
        }
    }
}
```

AWS CLI

此 AWS CLI 命令搜索基於輸入圖像集合中的用戶，與SearchUsersByImage操作。

```
aws rekognition search-users-by-image --image '{"S3Object":
{"Bucket":"s3BucketName","Name":"file-name"}}' --collection-id MyCollectionId --
region region-name
```

Python

下列範例會使用 SearchUsersByImage 操作，根據輸入映像搜尋集合中的使用者。

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

import boto3
from botocore.exceptions import ClientError
import logging
import os

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def load_image(file_name):
    """
    helper function to load the image for indexFaces call from local disk

    :param image_file_name: The image file location that will be used by
 indexFaces call.
    :return: The Image in bytes
    """
    print(f'- loading image: {file_name}')
    with open(file_name, 'rb') as file:
        return {'Bytes': file.read()}
```

```
def search_users_by_image(collection_id, image_file):
    """
    SearchUsersByImage operation with user ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param image_file: The image that contains the reference face to search
 for.

    :return: response of SearchUsersByImage API
    """
    logger.info(f'Searching for users using an image: {image_file}')
    try:
        response = client.search_users_by_image(
            CollectionId=collection_id,
            Image=load_image(image_file)
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsersByImage with given
 image: {image_file}')
        raise
    else:
        print(response)
        return response

def main():
    collection_id = "collection-id"
    IMAGE_SEARCH_SOURCE = os.getcwd() + '/image_path'
    search_users_by_image(collection_id, IMAGE_SEARCH_SOURCE)

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

SearchUsersByImage 操作請求

SearchUsersByImage 的請求包括要搜尋的集合和來源映像位置。在此範例中，來源映像儲存於 Amazon S3 儲存貯體 (S3Object)。另外指定要傳回的使用者最大臉數 (MaxUsers) 以及必須與要傳回的使用者相符的最小可信度 (UserMatchThreshold)。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsersByImage 作業回應

的回應SearchUsersByImage包括的FaceDetail物件SearchedFace，以及包 UserMatches 含UserIdSimilarity、和UserStatus的清單。如果輸入圖像包含多個面，則還 UnsearchedFaces 將返回一個列表。

```
{
    "SearchedFace": {
        "FaceDetail": {
            "BoundingBox": {
                "Width": 0.23692893981933594,
                "Top": 0.19235000014305115,
                "Left": 0.39177176356315613,
                "Height": 0.5437348484992981
            }
        }
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser1",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 100.0
        },
        {
            "User": {
```

```
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6",
    "UnsearchedFaces": [
        {
            "FaceDetails": {
                "BoundingBox": {
                    "Width": 0.031677018851041794,
                    "Top": 0.5593535900115967,
                    "Left": 0.6102562546730042,
                    "Height": 0.0682177022099495
                }
            },
            "Reasons": [
                "FACE_NOT_LARGEST"
            ]
        },
        {
            "FaceDetails": {
                "BoundingBox": {
                    "Width": 0.03254449740052223,
                    "Top": 0.6080358028411865,
                    "Left": 0.516062319278717,
                    "Height": 0.06347997486591339
                }
            },
            "Reasons": [
                "FACE_NOT_LARGEST"
            ]
        }
    ]
}
```


在儲存的影片中搜尋人臉

您可以在集合搜尋與儲存的影片或串流影片中偵測到的人物相符的人臉。此區段涵蓋在儲存的影片中搜尋人臉的操作。如需在串流影片搜尋人臉的相關資訊，請參閱 使用串流視訊事件。

您搜尋的臉孔必須先使用編製索引至集合中IndexFaces。如需詳細資訊，請參閱 新增人臉到集合。

Amazon Rekognition Video 人臉搜尋與分析儲存於 Amazon S3 儲存貯體中的影片的其它 Amazon Rekognition Video 操作一樣，採用相同的非同步工作流程。若要開始搜尋已儲存視訊中的臉孔，請撥打StartFaceSearch並提供您要搜尋之系列的 ID。Amazon Rekognition Video 會將影片分析的完成狀态发布至 Amazon Simple Notification Service (Amazon SNS) 主題。如果視頻分析成功，請致電GetFaceSearch以獲取搜索結果。如需開始影片分析並取得結果的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

下列程序說明如何在集合中搜尋與影片中偵測到的人物相符之人臉。程序也將說明如何取得影片中相符人物的追蹤資料。此程序會展開 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中的程式碼，該操作使用 Amazon Simple Queue Service (Amazon SQS) 佇列來取得影片分析要求的完成狀態。

搜尋與臉部相符的影片 (開發套件)

1. 建立集合。
2. 將臉部編入集合索引。
3. 執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。
4. 將下列程式碼新增至您在步驟 3 中建立的類別 `VideoDetect`。

   Java

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

      //Face collection search in video
 ======================================================================
      private static void StartFaceSearchCollection(String bucket, String
 video, String collection) throws Exception{

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

       StartFaceSearchRequest req = new StartFaceSearchRequest()
               .withCollectionId(collection)
               .withVideo(new Video()
                       .withS3Object(new S3Object()
                               .withBucket(bucket)
```

```
                                    .withName(video)))
                .withNotificationChannel(channel);



        StartFaceSearchResult startPersonCollectionSearchResult =
rek.startFaceSearch(req);
        startJobId=startPersonCollectionSearchResult.getJobId();

    }

    //Face collection search in video
======================================================================
    private static void GetFaceSearchCollectionResults() throws Exception{

       GetFaceSearchResult faceSearchResult=null;
       int maxResults=10;
       String paginationToken=null;

       do {

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


           faceSearchResult  = rek.getFaceSearch(
                   new GetFaceSearchRequest()
                   .withJobId(startJobId)
                   .withMaxResults(maxResults)
                   .withNextToken(paginationToken)
                   .withSortBy(FaceSearchSortBy.TIMESTAMP)
                   );


           VideoMetadata videoMetaData=faceSearchResult.getVideoMetadata();

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

```
          //Show search results
          List<PersonMatch> matches=
                  faceSearchResult.getPersons();

          for (PersonMatch match: matches) {
              long milliSeconds=match.getTimestamp();
              System.out.print("Timestamp: " + Long.toString(milliSeconds));
              System.out.println(" Person number: " +
match.getPerson().getIndex());
              List <FaceMatch> faceMatches = match.getFaceMatches();
              if (faceMatches != null) {
                  System.out.println("Matches in collection...");
                  for (FaceMatch faceMatch: faceMatches){
                      Face face=faceMatch.getFace();
                      System.out.println("Face Id: "+ face.getFaceId());
                      System.out.println("Similarity: " +
faceMatch.getSimilarity().toString());
                      System.out.println();
                  }
              }
              System.out.println();
          }

          System.out.println();

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

  }
```

在函數 main 中，將下行:

```
       StartLabelDetection(bucket, video);

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

取代為：

```
       String collection="collection";
       StartFaceSearchCollection(bucket, video, collection);
```

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

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;

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

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

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

```
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

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

        startFaceDetection(rekClient, channel, bucket, video);
        getFaceResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

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

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

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

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

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

    public static void getFaceResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetFaceDetectionResponse faceDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

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

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

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

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

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

```
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
 null.
                VideoMetadata videoMetaData =
 faceDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

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

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

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

Python

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

    # ============== Face Search ===============
    def StartFaceSearchCollection(self,collection):
```

```
        response = self.rek.start_face_search(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            CollectionId=collection,
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.snsTopicArn})

        self.startJobId=response['JobId']

        print('Start Job Id: ' + self.startJobId)


    def GetFaceSearchCollectionResults(self):
        maxResults = 10
        paginationToken = ''

        finished = False

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

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

            for personMatch in response['Persons']:
                print('Person Index: ' + str(personMatch['Person']['Index']))
                print('Timestamp: ' + str(personMatch['Timestamp']))

                if ('FaceMatches' in personMatch):
                    for faceMatch in personMatch['FaceMatches']:
                        print('Face ID: ' + faceMatch['Face']['FaceId'])
                        print('Similarity: ' + str(faceMatch['Similarity']))
                print()
            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
            print()
```

在函數 `main` 中，將下行:

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

取代為：

```
    collection='tests'
    analyzer.StartFaceSearchCollection(collection)

    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetFaceSearchCollectionResults()
```

如果已執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 以外的影片範例，要取代的程式碼可能會不同。

5. 將 `collection` 的值變更為您已在步驟 1 建立的集合名稱。
6. 執行程式碼。影片中的人物清單，而這些人物的人臉符合輸入集合中顯示的人臉。也將顯示每個符合的人物之追蹤資料。

GetFaceSearch 作業回應

以下是 `GetFaceSearch` 的 JSON 回應範例。

回應包含一系列的人物 (`Persons`)，這些在影片中偵測到的人物人臉符合輸入集合中的人臉。每次在視訊中符合人員時 PersonMatch，都會存在陣列元素。每個都`PersonMatch`包括來自輸入集合的臉部匹配的陣列 FaceMatch，有關匹配的人的信息 PersonDetail，以及該人在視頻中匹配的時間。

```
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "IJdbzkZfvBRqj8GPV82BPiZKkLOGCqDIsNZG/gQsEE5faTVK9JHOz/
xxxxxxxxxxxxxx",
    "Persons": [
        {
            "FaceMatches": [
                {
                    "Face": {
                        "BoundingBox": {
                            "Height": 0.527472972869873,
                            "Left": 0.33530598878860474,
```

```
                        "Top": 0.2161169946193695,
                        "Width": 0.35503000020980835
                    },
                    "Confidence": 99.90239715576172,
                    "ExternalImageId": "image.PNG",
                    "FaceId": "a2f2e224-bfaa-456c-b360-7c00241e5e2d",
                    "ImageId": "eb57ed44-8d8d-5ec5-90b8-6d190daff4c3"
                },
                "Similarity": 98.40909576416016
            }
        ],
        "Person": {
            "BoundingBox": {
                "Height": 0.8694444298744202,
                "Left": 0.2473958283662796,
                "Top": 0.10092592239379883,
                "Width": 0.49427083134651184
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.23000000417232513,
                    "Left": 0.42500001192092896,
                    "Top": 0.16333332657814026,
                    "Width": 0.12937499582767487
                },
                "Confidence": 99.97504425048828,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.46415066719055176,
                        "Y": 0.2572723925113678
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.5068183541297913,
                        "Y": 0.23705792427062988
                    },
                    {
                        "Type": "nose",
                        "X": 0.49765899777412415,
                        "Y": 0.28383663296699524
                    },
                    {
                        "Type": "mouthLeft",
```

```
                        "X": 0.487221896648407,
                        "Y": 0.3452930748462677
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.5142884850502014,
                        "Y": 0.33167609572410583
                    }
                ],
                "Pose": {
                    "Pitch": 15.966927528381348,
                    "Roll": -15.547388076782227,
                    "Yaw": 11.34195613861084
                },
                "Quality": {
                    "Brightness": 44.80223083496094,
                    "Sharpness": 99.95819854736328
                }
            },
            "Index": 0
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.2177777737379074,
                "Left": 0.7593749761581421,
                "Top": 0.13333334028720856,
                "Width": 0.12250000238418579
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.2177777737379074,
                    "Left": 0.7593749761581421,
                    "Top": 0.13333334028720856,
                    "Width": 0.12250000238418579
                },
                "Confidence": 99.63436889648438,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.8005779385566711,
                        "Y": 0.20915353298187256
```

```
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.8391435146331787,
                        "Y": 0.21049551665782928
                    },
                    {
                        "Type": "nose",
                        "X": 0.8191410899162292,
                        "Y": 0.2523227035999298
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.8093273043632507,
                        "Y": 0.29053622484207153
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.8366993069648743,
                        "Y": 0.29101791977882385
                    }
                ],
                "Pose": {
                    "Pitch": 3.165884017944336,
                    "Roll": 1.4182015657424927,
                    "Yaw": -11.151537895202637
                },
                "Quality": {
                    "Brightness": 28.910892486572266,
                    "Sharpness": 97.61507415771484
                }
            },
            "Index": 1
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.8388888835906982,
                "Left": 0,
                "Top": 0.15833333134651184,
                "Width": 0.2369791716337204
            },
```

```
                "Face": {
                    "BoundingBox": {
                        "Height": 0.20000000298023224,
                        "Left": 0.029999999329447746,
                        "Top": 0.2199999988079071,
                        "Width": 0.11249999701976776
                    },
                    "Confidence": 99.85971069335938,
                    "Landmarks": [
                        {
                            "Type": "eyeLeft",
                            "X": 0.06842322647571564,
                            "Y": 0.3010137975215912
                        },
                        {
                            "Type": "eyeRight",
                            "X": 0.10543643683195114,
                            "Y": 0.29697132110595703
                        },
                        {
                            "Type": "nose",
                            "X": 0.09569807350635529,
                            "Y": 0.33701086044311523
                        },
                        {
                            "Type": "mouthLeft",
                            "X": 0.0732642263174057,
                            "Y": 0.3757539987564087
                        },
                        {
                            "Type": "mouthRight",
                            "X": 0.10589495301246643,
                            "Y": 0.3722417950630188
                        }
                    ],
                    "Pose": {
                        "Pitch": -0.5589138865470886,
                        "Roll": -5.1093974113464355,
                        "Yaw": 18.69594955444336
                    },
                    "Quality": {
                        "Brightness": 43.052337646484375,
                        "Sharpness": 99.68138885498047
                    }
```

```
                },
                "Index": 2
            },
            "Timestamp": 0
        }......

    ],
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

在串流影片中搜尋集合中的人臉

您可以使用 Amazon Rekognition Video 來偵測及辨識串流影片中的人臉。使用 Amazon Rekognition Video，您可以建立串流處理器 (CreateStreamProcessor) 以開始和管理串流視訊的分析。

為了偵測影片串流中的已知人臉 (人臉搜尋)，Amazon Rekognition Video 使用 Amazon Kinesis Video Streams 來接收和處理影片串流。分析結果會從 Amazon Rekognition Video 輸出至 Kinesis 資料串流，再由您的使用者端應用程式讀取。

若要搭配串流影片使用 Amazon Rekognition Video，您的應用程式需要實作下列專案：

- Kinesis video stream，用於將串流影片傳送至 Amazon Rekognition Video。如需詳細資訊，請參閱 Amazon Kinesis Video Streams 開發人員指南。
- Amazon Rekognition Video 串流處理器用於管理串流影片的分析。如需詳細資訊，請參閱 亞馬遜視訊串流處理器操作概觀。
- Kinesis 資料串流消費者可讀取 Amazon Rekognition Video 傳送至 Kinesis 資料串流的分析結果。如需詳細資訊，請參閱 Kinesis 資料串流消費者。

本節包含撰寫建立 Kinesis 影片串流和其他必要資源的應用程式、將影片串流至 Amazon Rekognition Video，以及接收分析結果的相關資訊。

主題

設定您的 Amazon Rekognition Video 和 Amazon Kinesis 資源

下列程序說明佈建 Kinesis 影片串流所採取的步驟，以及用來辨識串流影片中人臉的其他資源。

必要條件

要運行此過程，你需要安 AWS SDK for Java 裝。如需詳細資訊，請參閱 Amazon Rekognition 入門。AWS 帳戶 您使用的項目必須具有 Amazon Rekognition API 的存取權限。如需詳細資訊，請參閱 IAM 使用者指南中的 Amazon Rekognition 定義的動作。

辨識影片串流中的臉部 (AWS 開發套件)

1. 如果您尚未建立 IAM 服務角色，請建立 IAM 服務角色，讓 Amazon Rekognition Video 有權存取您的 Kinesis 影片串流和 Kinesis 資料串流。請記下 ARN。如需詳細資訊，請參閱 使用以下方式存取串流 AmazonRekognitionServiceRole。
2. 建立集合並記下您所使用的集合識別符。
3. 將您要搜尋之臉部的索引建立至您於步驟 2 中建立的集合。
4. 建立 Kinesis 資料串流，並記下串流的 Amazon Resource Name (ARN)。
5. 建立 Kinesis 資料串流。在串流名稱前加上，AmazonRekognition並記下串流的 ARN。

然後，您可以建立人臉搜尋串流處理器，並使用您選擇的串流處理器名稱啟動串流處理器。

> **Note**
>
> 只有在驗證可以將媒體匯入 Kinesis 影片串流後，才應啟動串流處理器。

將影片串流式傳輸到 Amazon Rekognition Video

若要將影片串流傳輸至 Amazon Rekognition Video，您可以使用 Amazon Kinesis Video Streams SDK 來建立和使用 Kinesis 影片串流。`PutMedia` 操作會將影片資料片段寫入 Amazon Rekognition Video 所取用的 Kinesis 影片串流。每個影片資料片段的長度通常是 2 至 10 秒，並包含一連串獨立的影片影

格。Amazon Rekognition Video 支援 H.264 編碼影片，包含三種影格類型 (I、B 與 P)。如需詳細資訊，請參閱 I 影格。片段中的第一個影格必須是 I 影格。I 影格可以與任何其他影格分開解碼。

當影片資料進入 Kinesis 影片串流時，會指派唯一號碼給片段。如需範例，請參閱 PutMedia API 範例。

- 如果您是從 Matroska (MKV) 編碼來源串流，請使用此PutMedia作業將來源視訊串流至您建立的 Kinesis 視訊串流。如需詳細資訊，請參閱 PutMedia API 範例。
- 如果您是從裝置相機獲得串流，請參閱 使用 GStreamer 外掛程式進行串流。

讓 Amazon Rekognition Video 有權存取您的資源

您可以使用 AWS Identity and Access Management (IAM) 服務角色，將 Kinesis 影片串流的 Amazon Rekognition Video 讀取存取權授予。如果您使用的是人臉搜尋串流處理器，則可以使用 IAM 服務角色讓 Amazon Rekognition Video 有權寫入 Kinesis 資料串流。如果您使用的是安全監控串流處理器，則可以使用 IAM 角色讓 Amazon Rekognition Video 有權存取 Amazon S3 儲存貯體和 Amazon SNS 主題。

提供人臉搜尋串流處理器的存取權

您可以建立許可政策，其允許 Amazon Rekognition Video 對 Kinesis 影片串流和 Kinesis 資料串流的存取。

對 Amazon Rekognition Video 提供人臉搜尋串流處理器的存取權

1. 使用 IAM JSON 政策編輯器建立新的許可政策，並使用下列政策。以所需的 Kinesis 影片串流 ARN 取代 `video-arn`。如果您使用的是人臉搜尋串流處理器，請以所需的 Kinesis 資料串流 ARN 取代 `data-arn`。

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

```
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": "video-arn"
        }
    ]
}
```

2. 建立 IAM 服務角色，或更新現有的 IAM 服務角色。請使用下列資訊來建立 IAM 服務角色：

    1. 針對服務名稱選擇 Rekognition。
    2. 為服務角色使用案例選擇 Rekognition。
    3. 連接您在步驟 1 中建立的許可政策。
3. 請記下服務角色的 ARN。您需要此資訊才可開始影片分析操作。

使用以下方式存取串流 AmazonRekognitionServiceRole

作為設定 Kinesis 影片串流和資料串流存取權的替代選項，可以使用 `AmazonRekognitionServiceRole` 许可政策。IAM 提供 Rekognition 服務角色使用案例，配合 `AmazonRekognitionServiceRole` 許可政策使用時，擁有對多個 Kinesis 資料串流的寫入權和對所有 Kinesis 影片串流的讀取權。若要讓 Amazon Rekognition Video 寫入存取多個 Kinesis 資料串流，您可以在 Kinesis 資料串流的名稱前面加上 — 例如。AmazonRekognition`AmazonRekognitionMyDataStreamName`

讓 Amazon Rekognition Video 有權存取您的 Kinesis 影片串流和 Kinesis 資料串流

1. 建立 IAM 服務角色。請使用下列資訊來建立 IAM 服務角色：

    1. 針對服務名稱選擇 Rekognition。
    2. 為服務角色使用案例選擇 Rekognition。
    3. 選擇AmazonRekognitionServiceRole許可政策，該政策可讓 Amazon Rekognition Video 寫入存取 Kinesis 資料串流，這些資料串流的前置詞為您的所有 Kinesis 影片串流，以AmazonRekognition及對所有 Kinesis 影片串流的讀取存取權。
2. 為了確保您 AWS 帳戶 的安全，請將 Rekognition 的存取範圍限制在您正在使用的資源。這可以通過將信任政策附加到您的 IAM 服務角色來完成。如需如何執行此作業的資訊，請參閱 預防跨服務混淆代理人。

3. 記下服務角色的 Amazon Resource Name (ARN)。您需要此資訊才可開始影片分析操作。

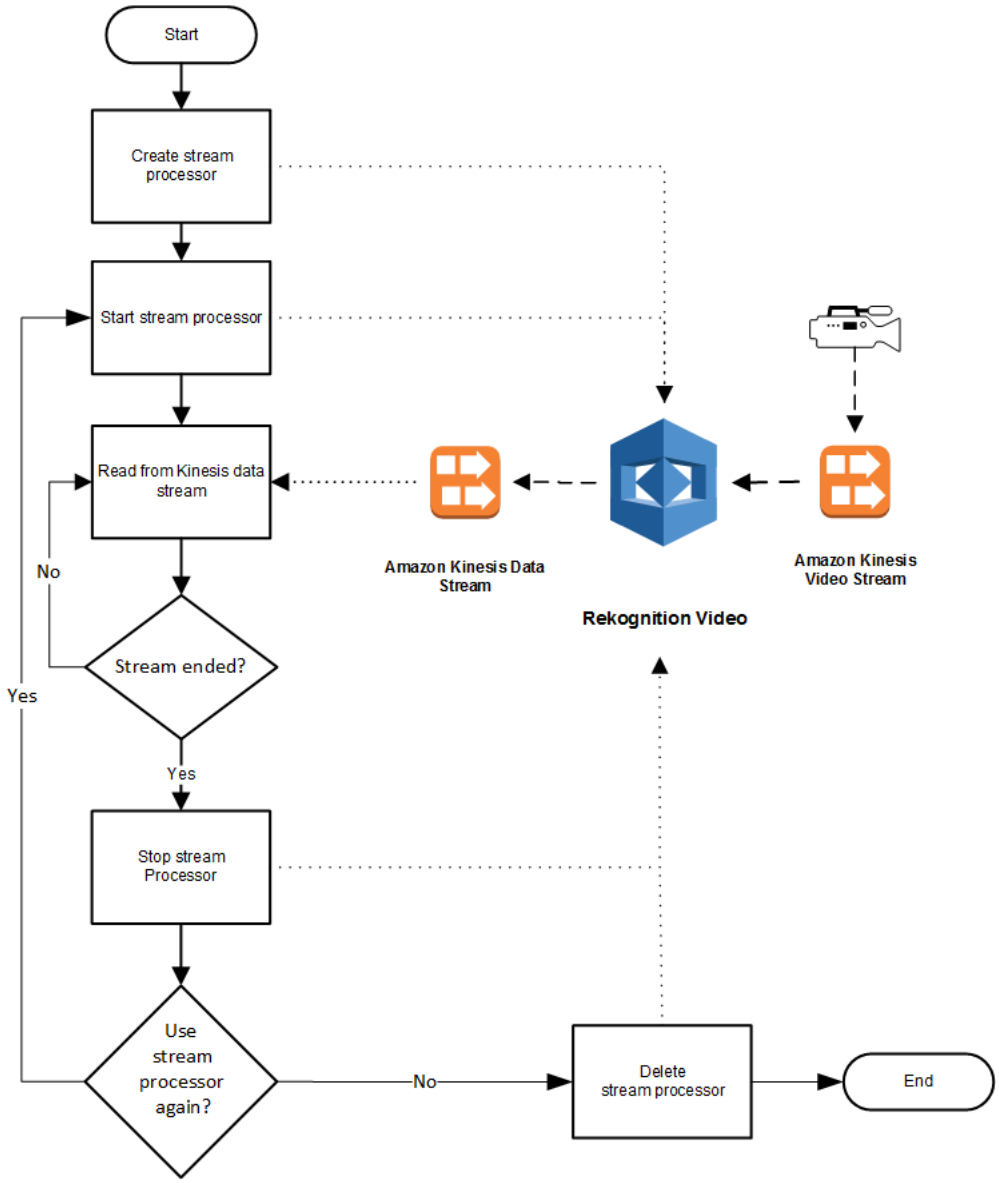
在串流影片中搜尋人臉。

Amazon Rekognition Video 可在集合中搜尋與串流影片中偵測到之人臉相符的人臉。如需集合的詳細資訊，請參閱 在集合中搜尋人臉。

主題

- 建立 Amazon Rekognition Video 人臉搜尋串流處理器
- 啟動 Amazon Rekognition Video 人臉搜尋串流處理器
- 使用串流處理器進行人臉搜尋 (Java V2 範例)
- 使用串流處理器進行人臉搜尋 (Java V1 範例)
- 讀取串流影片分析結果
- 參考：Kinesis 人臉辨識記錄

下圖顯示 Amazon Rekognition Video 如何偵測及辨識串流影片中的人臉。



建立 Amazon Rekognition Video 人臉搜尋串流處理器

您必須先建立 Amazon Rekognition 視訊串流處理器 ()，才能分析串流影片。CreateStreamProcessor串流處理器包含 Kinesis 資料串流和 Kinesis 影片串流的相關資訊。其也會包含集合的識別符，該集合含有您要在輸入串流影片中辨識的人臉。您也可以指定串流處理器的名稱。以下是 `CreateStreamProcessor` 要求的 JSON 範例。

```
{
	"Name": "streamProcessorForCam",
	"Input": {
		"KinesisVideoStream": {
```

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

以下是 CreateStreamProcessor 的回應範例。

```
{
        “StreamProcessorArn”: “arn:aws:rekognition:us-
east-1:nnnnnnnnnnnn:streamprocessor/streamProcessorForCam”
}
```

啟動 Amazon Rekognition Video 人臉搜尋串流處理器

您可以StartStreamProcessor使用您在中指定的串流處理器名稱呼叫來開始分析串流視訊CreateStreamProcessor。以下是 StartStreamProcessor 要求的 JSON 範例。

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

如果串流處理器成功啟動，就會傳回 HTTP 200 回應，以及空的 JSON 內文。

使用串流處理器進行人臉搜尋 (Java V2 範例)

下列範例程式碼示範如何呼叫各種串流處理器作業 StartStreamProcessor，例如CreateStreamProcessor和，使用 Java 版本 2 的 AWS SDK。

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorRequest;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorResponse;
import software.amazon.awssdk.services.rekognition.model.FaceSearchSettings;
import software.amazon.awssdk.services.rekognition.model.KinesisDataStream;
import software.amazon.awssdk.services.rekognition.model.KinesisVideoStream;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsRequest;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.StreamProcessor;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorInput;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorSettings;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorOutput;
import software.amazon.awssdk.services.rekognition.model.StartStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorResponse;

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

                                Usage:    <role> <kinInputStream> <kinOutputStream>
 <collectionName> <StreamProcessorName>

                                Where:
                                   role - The ARN of the AWS Identity and Access
 Management (IAM) role to use. \s
                                   kinInputStream - The ARN of the Kinesis video
 stream.\s
                                   kinOutputStream - The ARN of the Kinesis data
 stream.\s
```

```
                                collectionName - The name of the collection to use
that contains content. \s
                                StreamProcessorName - The name of the Stream
Processor. \s
                             """;

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

                String role = args[0];
                String kinInputStream = args[1];
                String kinOutputStream = args[2];
                String collectionName = args[3];
                String streamProcessorName = args[4];

                Region region = Region.US_EAST_1;
                RekognitionClient rekClient = RekognitionClient.builder()
                                .region(region)
                                .build();

                processCollection(rekClient, streamProcessorName, kinInputStream,
kinOutputStream, collectionName,
                                role);
                startSpecificStreamProcessor(rekClient, streamProcessorName);
                listStreamProcessors(rekClient);
                describeStreamProcessor(rekClient, streamProcessorName);
                deleteSpecificStreamProcessor(rekClient, streamProcessorName);
        }

        public static void listStreamProcessors(RekognitionClient rekClient) {
                ListStreamProcessorsRequest request =
ListStreamProcessorsRequest.builder()
                                .maxResults(15)
                                .build();

                ListStreamProcessorsResponse listStreamProcessorsResult =
rekClient.listStreamProcessors(request);
                for (StreamProcessor streamProcessor :
listStreamProcessorsResult.streamProcessors()) {
                        System.out.println("StreamProcessor name - " +
streamProcessor.name());
                        System.out.println("Status - " + streamProcessor.status());
```

```
                }
        }

        private static void describeStreamProcessor(RekognitionClient rekClient, String
StreamProcessorName) {
                DescribeStreamProcessorRequest streamProcessorRequest =
DescribeStreamProcessorRequest.builder()
                                .name(StreamProcessorName)
                                .build();

                DescribeStreamProcessorResponse describeStreamProcessorResult =
rekClient
                                .describeStreamProcessor(streamProcessorRequest);
                System.out.println("Arn - " +
describeStreamProcessorResult.streamProcessorArn());
                System.out.println("Input kinesisVideo stream - "
                                +
describeStreamProcessorResult.input().kinesisVideoStream().arn());
                System.out.println("Output kinesisData stream - "
                                +
describeStreamProcessorResult.output().kinesisDataStream().arn());
                System.out.println("RoleArn - " +
describeStreamProcessorResult.roleArn());
                System.out.println(
                                "CollectionId - "
                                                +
describeStreamProcessorResult.settings().faceSearch().collectionId());
                System.out.println("Status - " +
describeStreamProcessorResult.status());
                System.out.println("Status message - " +
describeStreamProcessorResult.statusMessage());
                System.out.println("Creation timestamp - " +
describeStreamProcessorResult.creationTimestamp());
                System.out.println("Last update timestamp - " +
describeStreamProcessorResult.lastUpdateTimestamp());
        }

        private static void startSpecificStreamProcessor(RekognitionClient rekClient,
String StreamProcessorName) {
                try {
                        StartStreamProcessorRequest streamProcessorRequest =
StartStreamProcessorRequest.builder()
                                        .name(StreamProcessorName)
                                        .build();
```

```
                        rekClient.startStreamProcessor(streamProcessorRequest);
                        System.out.println("Stream Processor " + StreamProcessorName +
" started.");

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

        private static void processCollection(RekognitionClient rekClient, String
StreamProcessorName,
                        String kinInputStream, String kinOutputStream, String
collectionName, String role) {
                try {
                        KinesisVideoStream videoStream = KinesisVideoStream.builder()
                                        .arn(kinInputStream)
                                        .build();

                        KinesisDataStream dataStream = KinesisDataStream.builder()
                                        .arn(kinOutputStream)
                                        .build();

                        StreamProcessorOutput processorOutput =
StreamProcessorOutput.builder()
                                        .kinesisDataStream(dataStream)
                                        .build();

                        StreamProcessorInput processorInput =
StreamProcessorInput.builder()
                                        .kinesisVideoStream(videoStream)
                                        .build();

                        FaceSearchSettings searchSettings =
FaceSearchSettings.builder()
                                        .faceMatchThreshold(75f)
                                        .collectionId(collectionName)
                                        .build();

                        StreamProcessorSettings processorSettings =
StreamProcessorSettings.builder()
                                        .faceSearch(searchSettings)
                                        .build();
```

```
                        CreateStreamProcessorRequest processorRequest =
 CreateStreamProcessorRequest.builder()
                                        .name(StreamProcessorName)
                                        .input(processorInput)
                                        .output(processorOutput)
                                        .roleArn(role)
                                        .settings(processorSettings)
                                        .build();

                        CreateStreamProcessorResponse response =
 rekClient.createStreamProcessor(processorRequest);
                        System.out.println("The ARN for the newly create stream
 processor is "
                                        + response.streamProcessorArn());

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

        private static void deleteSpecificStreamProcessor(RekognitionClient rekClient,
 String StreamProcessorName) {
                rekClient.stopStreamProcessor(a -> a.name(StreamProcessorName));
                rekClient.deleteStreamProcessor(a -> a.name(StreamProcessorName));
                System.out.println("Stream Processor " + StreamProcessorName + "
 deleted.");
        }
}
```

使用串流處理器進行人臉搜尋 (Java V1 範例)

下列範例程式碼示範如何呼叫各種串流處理器作業，例如CreateStreamProcessor和 StartStreamProcessor，使用 Java V1。此範例包含串流處理器管理員 class (StreamManager)，該類別提供呼叫串流處理器作業的方法。入門類（Starter）創建一個 StreamManager 對象並調用各種操作。

設定範例：

1. 設定數值的 Starter 類別成員欄位至您所需的數值。
2. 在 Starter 類別函數 `main`，移除所需的函數呼叫。

Starter 類別

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

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

public class Starter {

 public static void main(String[] args) {


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

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

StreamManager 類

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

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

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CreateStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.CreateStreamProcessorResult;
import com.amazonaws.services.rekognition.model.DeleteStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.DeleteStreamProcessorResult;
import com.amazonaws.services.rekognition.model.DescribeStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.DescribeStreamProcessorResult;
import com.amazonaws.services.rekognition.model.FaceSearchSettings;
import com.amazonaws.services.rekognition.model.KinesisDataStream;
import com.amazonaws.services.rekognition.model.KinesisVideoStream;
import com.amazonaws.services.rekognition.model.ListStreamProcessorsRequest;
import com.amazonaws.services.rekognition.model.ListStreamProcessorsResult;
import com.amazonaws.services.rekognition.model.StartStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.StartStreamProcessorResult;
import com.amazonaws.services.rekognition.model.StopStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.StopStreamProcessorResult;
import com.amazonaws.services.rekognition.model.StreamProcessor;
import com.amazonaws.services.rekognition.model.StreamProcessorInput;
import com.amazonaws.services.rekognition.model.StreamProcessorOutput;
import com.amazonaws.services.rekognition.model.StreamProcessorSettings;

public class StreamManager {

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

```
    private AmazonRekognition rekognitionClient;


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

    }

    public void createStreamProcessor() {
     //Setup input parameters
        KinesisVideoStream kinesisVideoStream = new
KinesisVideoStream().withArn(kinesisVideoStreamArn);
        StreamProcessorInput streamProcessorInput =
                new StreamProcessorInput().withKinesisVideoStream(kinesisVideoStream);
        KinesisDataStream kinesisDataStream = new
KinesisDataStream().withArn(kinesisDataStreamArn);
        StreamProcessorOutput streamProcessorOutput =
                new StreamProcessorOutput().withKinesisDataStream(kinesisDataStream);
        FaceSearchSettings faceSearchSettings =
                new
FaceSearchSettings().withCollectionId(collectionId).withFaceMatchThreshold(matchThreshold);
        StreamProcessorSettings streamProcessorSettings =
                new StreamProcessorSettings().withFaceSearch(faceSearchSettings);

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

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

        //Display result
```

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

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

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

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

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

        //Display various stream processor attributes.
        System.out.println("Arn - " +
describeStreamProcessorResult.getStreamProcessorArn());
        System.out.println("Input kinesisVideo stream - "
                +
describeStreamProcessorResult.getInput().getKinesisVideoStream().getArn());
        System.out.println("Output kinesisData stream - "
                +
describeStreamProcessorResult.getOutput().getKinesisDataStream().getArn());
        System.out.println("RoleArn - " + describeStreamProcessorResult.getRoleArn());
        System.out.println(
                "CollectionId - " +
describeStreamProcessorResult.getSettings().getFaceSearch().getCollectionId());
```

```
        System.out.println("Status - " + describeStreamProcessorResult.getStatus());
        System.out.println("Status message - " +
 describeStreamProcessorResult.getStatusMessage());
        System.out.println("Creation timestamp - " +
 describeStreamProcessorResult.getCreationTimestamp());
        System.out.println("Last update timestamp - " +
 describeStreamProcessorResult.getLastUpdateTimestamp());
    }

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

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

讀取串流影片分析結果

您可以使用 Amazon Kinesis Data Streams Client Library 取用傳送至 Amazon Kinesis Data Streams 輸出串流的分析結果。如需詳細資訊，請參閱從 Kinesis 資料串流讀取資料。Amazon Rekognition Video 會將每個已分析影格的 JSON 影格記錄放入 Kinesis 輸出串流。Amazon Rekognition Video 不會分析透過 Kinesis 影片串流傳遞給其的每個影格。

傳送至 Kinesis 資料串流的影格記錄包含有關影格所在之 Kinesis 影片串流片段、影格在片段中的位置，以及影格中已辨識之人臉的資訊。它也包含串流處理器的狀態資訊。如需詳細資訊，請參閱 參考：Kinesis 人臉辨識記錄。

Amazon Kinesis Video Streams 剖析器程式庫包含範例測試，這些測試會使用 Amazon Rekognition Video 結果，並將其與原始 Kinesis 影片串流整合。如需詳細資訊，請參閱 使用 Kinesis Video Streams 本機顯示 Rekognition 結果。

Amazon Rekognition Video 訊將 Amazon Rekognition Video 訊分析資訊串流到 Kinesis 資料串流。以下是單一記錄的 JSON 範例。

```
{
```

```
  "InputInformation": {
    "KinesisVideo": {
      "StreamArn": "arn:aws:kinesisvideo:us-west-2:nnnnnnnnnnnn:stream/stream-name",
      "FragmentNumber": "91343852333289682796718532614445757584843717598",
      "ServerTimestamp": 1510552593.455,
      "ProducerTimestamp": 1510552593.193,
      "FrameOffsetInSeconds": 2
    }
  },
  "StreamProcessorInformation": {
    "Status": "RUNNING"
  },
  "FaceSearchResponse": [
    {
      "DetectedFace": {
        "BoundingBox": {
          "Height": 0.075,
          "Width": 0.05625,
          "Left": 0.428125,
          "Top": 0.40833333
        },
        "Confidence": 99.975174,
        "Landmarks": [
          {
            "X": 0.4452057,
            "Y": 0.4395594,
            "Type": "eyeLeft"
          },
          {
            "X": 0.46340984,
            "Y": 0.43744427,
            "Type": "eyeRight"
          },
          {
            "X": 0.45960626,
            "Y": 0.4526856,
            "Type": "nose"
          },
          {
            "X": 0.44958648,
            "Y": 0.4696949,
            "Type": "mouthLeft"
          },
          {
```

```
            "X": 0.46409217,
            "Y": 0.46704912,
            "Type": "mouthRight"
          }
        ],
        "Pose": {
          "Pitch": 2.9691637,
          "Roll": -6.8904796,
          "Yaw": 23.84388
        },
        "Quality": {
          "Brightness": 40.592964,
          "Sharpness": 96.09616
        }
      },
      "MatchedFaces": [
        {
          "Similarity": 88.863960,
          "Face": {
            "BoundingBox": {
              "Height": 0.557692,
              "Width": 0.749838,
              "Left": 0.103426,
              "Top": 0.206731
            },
            "FaceId": "ed1b560f-d6af-5158-989a-ff586c931545",
            "Confidence": 99.999201,
            "ImageId": "70e09693-2114-57e1-807c-50b6d61fa4dc",
            "ExternalImageId": "matchedImage.jpeg"
          }
        }
      ]
    }
  ]
}
```

在 JSON 範例中，請注意下列事項：

- InputInformation— 有關用於將視訊串流至亞馬遜 Rekognition 視訊的 Kinesis 視訊串流的資訊。如需詳細資訊，請參閱 InputInformation。
- StreamProcessorInformation— 亞馬遜視訊串流處理器的狀態資訊。唯一可用於 `Status` 欄位的值是 RUNNING。如需詳細資訊，請參閱 StreamProcessorInformation。

- FaceSearchResponse— 包含串流視訊中符合輸入集合中人臉的臉孔的相關資訊。FaceSearchResponse包含一個DetectedFace對象，該對象是在分析的視頻幀中檢測到的臉部。對於每個偵測到的人臉，陣列 MatchedFaces 皆包含在輸入集合中找到之相符人臉物件的陣列 (MatchedFace)，以及相似度分數。

將 Kinesis 影片串流對應至 Kinesis 資料串流

您可能會想要將 Kinesis 影片串流影格對應至已發送至 Kinesis 資料串流的影格。例如，在串流影片顯示期間，您可能想要在已辨識的人臉周圍顯示方塊。週框方塊座標會做為 Kinesis 人臉辨識記錄的一部分傳送至 Kinesis 資料串流。若要正確顯示週框方塊，您需要將隨 Kinesis 人臉辨識記錄傳送的時間資訊，與來源 Kinesis 影片串流中相應的影格相對應。

您用來將 Kinesis 影片串流對應至 Kinesis 資料串流的技術取決於您要串流即時媒體 (例如即時串流影片)，還是要串流封存媒體 (例如已儲存影片)。

串流即時媒體時的對應

若要將 Kinesis 影片串流影格對應至 Kinesis 資料串流影格

1. 將作業的輸入參FragmentTimeCodeType數設PutMedia定為RELATIVE。
2. 呼叫 PutMedia，將即時媒體傳遞至 Kinesis 影片串流。
3. 當您從 Kinesis 資料串流收到 Kinesis 人臉辨識記錄時，請儲存來自 KinesisVideo 欄位中 ProducerTimestamp 與 FrameOffsetInSeconds 的值。
4. 將 ProducerTimestamp 與 FrameOffsetInSeconds 欄位值相加，藉此計算對應至 Kinesis 影片串流影格的時間戳記。

串流封存媒體時的對應

若要將 Kinesis 影片串流影格對應至 Kinesis 資料串流影格

1. 呼叫PutMedia將封存的媒體傳遞至 Kinesis 視訊串流。
2. 當您從 PutMedia 操作回應收到 Acknowledgement 物件時，請儲存承載欄位中的 FragmentNumber 欄位值。FragmentNumber 是 MKV 叢集的片段數目。
3. 當您從 Kinesis 資料串流收到 Kinesis 人臉辨識記錄時，請儲存來自 KinesisVideo 欄位的 FrameOffsetInSeconds 欄位值。
4. 使用您在步驟 2 與 3 中儲存的 FrameOffsetInSeconds 與 FragmentNumber 值來計算對應。FrameOffsetInSeconds 是具有特定 FragmentNumber 之片段的位移，其會傳送至

Amazon Kinesis Data Streams。如需取得指定片段號碼之影片影格的詳細資訊，請參閱 Amazon Kinesis Video Streams 的已封存媒體。

使用 Kinesis Video Streams 本機顯示 Rekognition 結果

您可以使用 Amazon Kinesis 影片串流剖析器程式庫的範例測試範例測試，在 Amazon Kinesis Video Streams 中查看 Amazon Rekognition 影片的結果-重新認知範例。KinesisVideo KinesisVideoRekognitionIntegrationExample 會在偵測到的人臉上顯示邊界方框，並透過 JFrame 在本機上轉譯影片。此程序假設您已成功將媒體輸入從裝置攝影機連接到 Kinesis 影片串流，並啟動 Amazon Rekognition 串流處理器。如需詳細資訊，請參閱 使用 GStreamer 外掛程式進行串流。

步驟 1：安裝 Kinesis 影片串流剖析程式庫

若要建立目錄並下載 Github 儲存庫，請執行下列命令：

```
$ git clone https://github.com/aws/amazon-kinesis-video-streams-parser-library.git
```

導航到庫目錄並執行以下 Maven 命令以執行全新安裝：

```
$ mvn clean install
```

步驟 2：設定 Kinesis 影片串流和 Rekognition 整合範例測試

開啟 KinesisVideoRekognitionIntegrationExampleTest.java 檔案。刪除類標題後的 @Ignore 權利。使用來自 Amazon Kinesis 和 Amazon Rekognition 資源的資訊填入資料欄位。如需詳細資訊，請參閱 設定您的 Amazon Rekognition Video 和 Amazon Kinesis 資源。如果您要將影片串流至 Kinesis 影片串流，請移除 inputStream 參數。

請參閱以下程式碼範例。

```
RekognitionInput rekognitionInput = RekognitionInput.builder()
  .kinesisVideoStreamArn("arn:aws:kinesisvideo:us-east-1:123456789012:stream/
rekognition-test-video-stream")
  .kinesisDataStreamArn("arn:aws:kinesis:us-east-1:123456789012:stream/
AmazonRekognition-rekognition-test-data-stream")
  .streamingProcessorName("rekognition-test-stream-processor")
  // Refer how to add face collection :
```

```
  // https://docs.aws.amazon.com/rekognition/latest/dg/add-faces-to-collection-
procedure.html
  .faceCollectionId("rekognition-test-face-collection")
  .iamRoleArn("rekognition-test-IAM-role")
  .matchThreshold(0.95f)
  .build();

KinesisVideoRekognitionIntegrationExample example =
 KinesisVideoRekognitionIntegrationExample.builder()
  .region(Regions.US_EAST_1)
  .kvsStreamName("rekognition-test-video-stream")
  .kdsStreamName("AmazonRekognition-rekognition-test-data-stream")
  .rekognitionInput(rekognitionInput)
  .credentialsProvider(new ProfileCredentialsProvider())
  // NOTE: Comment out or delete the inputStream parameter if you are streaming video,
 otherwise
  // the test will use a sample video.
  //.inputStream(TestResourceUtil.getTestInputStream("bezos_vogels.mkv"))
  .build();
```

步驟 3：執行 Kinesis Video Streams 和 Rekognition 整合範例測試

如果您要串流到 Kinesis 影片串流，請確保 Kinesis 影片串流正在接收媒體輸入，並在執行 Amazon Rekognition Video 串流處理器的情況下開始分析串流。如需詳細資訊，請參閱 亞馬遜視訊串流處理器操作概觀。執行該 `KinesisVideoRekognitionIntegrationExampleTest` 類作為一個 JUnit 測試。短暫的延遲後，會開啟一個新視窗，其中包含 Kinesis 影片串流中的影片，並在偵測到的人臉上繪製邊框。

> **Note**
>
> 此範例中使用的集合中的臉孔必須具有以此格式指定的外部影像 ID (檔案名稱)，才能讓邊界方框標籤顯示有意義的文字： PersonName1-信任、 PersonName 2 入侵者、 PersonName 3-中立等。標籤也可以進行顏色編碼，並且可以在 FaceType .java 文件中進行自定義。

參考：Kinesis 人臉辨識記錄

您可以使用 Amazon Rekognition Video 來辨識串流影片中的人臉。對於每個已分析的影格，Amazon Rekognition Video 都會將 JSON 影格記錄輸出至 Kinesis 資料串流。Amazon Rekognition Video 不會分析透過 Kinesis 影片串流傳遞給其的每個影格。

JSON 影格記錄包含輸入與輸出串流的資訊、串流處理器的狀態，以及已分析影格中所辨識之臉部的資訊。本節包含 JSON 影格記錄的參考資訊。

以下是適用於 Kinesis 資料串流記錄的 JSON 語法。如需詳細資訊，請參閱 使用串流視訊事件。

> **Note**
>
> Amazon Rekognition Video API 的運作原理是將輸入流中的人臉與一組人臉進行比較，並傳回最接近找到的符合專案以及相似性分數。

```
{
    "InputInformation": {
        "KinesisVideo": {
            "StreamArn": "string",
            "FragmentNumber": "string",
            "ProducerTimestamp": number,
            "ServerTimestamp": number,
            "FrameOffsetInSeconds": number
        }
    },
    "StreamProcessorInformation": {
        "Status": "RUNNING"
    },
    "FaceSearchResponse": [
        {
            "DetectedFace": {
                "BoundingBox": {
                    "Width": number,
                    "Top": number,
                    "Height": number,
                    "Left": number
                },
                "Confidence": number,
                "Landmarks": [
                    {
                        "Type": "string",
                        "X": number,
                        "Y": number
                    }
                ],
                "Pose": {
```

```
                    "Pitch": number,
                    "Roll": number,
                    "Yaw": number
                },
                "Quality": {
                    "Brightness": number,
                    "Sharpness": number
                }
            },
            "MatchedFaces": [
                {
                    "Similarity": number,
                    "Face": {
                        "BoundingBox": {
                            "Width": number,
                            "Top": number,
                            "Height": number,
                            "Left": number
                        },
                        "Confidence": number,
                        "ExternalImageId": "string",
                        "FaceId": "string",
                        "ImageId": "string"
                    }
                }
            ]
        }
    ]
}
```

JSON 記錄

JSON 記錄包含 Amazon Rekognition Video 所處理之影格的資訊。此記錄包括串流影片的資訊、已分析影格的狀態，以及影格中所辨識之臉部的資訊。

InputInformation

有關用於將影片串流至 Amazon Rekognition Video 的 Kinesis 影片串流的相關資訊。

類型：InputInformation 物件

StreamProcessorInformation

Amazon Rekognition Video 串流處理器的資訊。這包括串流處理器目前狀態的狀態資訊。

類型：StreamProcessorInformation 物件

FaceSearchResponse

關於串流影片影格內偵測到的臉部與輸入集合中之相符臉部的資訊。

類型：FaceSearchResponse 物件陣列

InputInformation

Amazon Rekognition Video 使用之來源影片串流的資訊。如需詳細資訊，請參閱 使用串流視訊事件。

KinesisVideo

類型：KinesisVideo 物件

KinesisVideo

將來源影片匯入 Amazon Rekognition Video 的 Kinesis 影片串流的相關資訊。如需詳細資訊，請參閱使用串流視訊事件。

StreamArn

Kinesis 資料串流的 Amazon Resource Name (ARN)。

類型：字串

FragmentNumber

包含此記錄代表之影格的串流影片片段。

類型：字串

ProducerTimestamp

片段的生產者端 Unix 時間戳記。如需詳細資訊，請參閱PutMedia。

類型：數字

ServerTimestamp

片段的伺服器端 Unix 時間戳記。如需詳細資訊，請參閱PutMedia。

類型：數字

FrameOffsetInSeconds

片段內影格的位移 (以秒為單位)。

類型：數字

StreamProcessorInformation

串流處理器的狀態資訊。

狀態

串流處理器目前的狀態。唯一可能的值為「RUNNING」。

類型：字串

FaceSearchResponse

關於串流影片影格內偵測到的臉部以及輸入集合中與偵測到的臉部相符之臉部的資訊。您可以在呼叫中指定集合CreateStreamProcessor。如需詳細資訊，請參閱 使用串流視訊事件。

DetectedFace

在用於分析的影片影格中偵測到的人臉之人臉詳細資訊。

類型：DetectedFace 物件

MatchedFaces

集合中的臉部之臉部詳細資訊陣列，符合於 `DetectedFace` 中偵測到的臉部。

類型：MatchedFace 物件陣列

DetectedFace

串流影片影格中偵測到的臉部之資訊。輸入集合中相符的臉部可於 MatchedFace 物件欄位中找到。

BoundingBox

在已分析影片影格中偵測到之臉部的週框方塊座標。 BoundingBox 物件具有與用於影像分析的 BoundingBox 物件相同的性質。

類型：BoundingBox 物件

可信度

Amazon Rekognition Video 偵測到的人臉實際上是人臉的可信度級別 (1-100) 。1 是最低的可信度，100 是最高的可信度。

類型：數字

特徵點

臉部特徵點的陣列。

類型：特徵點物件陣列

姿態

表示由俯仰、側傾與偏轉所決定的臉部姿態。

類型：姿態物件

品質

識別臉部影像的亮度與銳利度。

類型：ImageQuality 物件

MatchedFace

用於分析的影片影格中偵測到的相符人臉之資訊。

臉部

輸入集合內與 DetectedFace 物件中臉部相符之臉部的臉部配對資訊。

類型：人臉物件

相似度

人臉相符的可信度 (1-100)。1 是最低的可信度，100 是最高的可信度。

類型：數字

使用 GStreamer 外掛程式進行串流

Amazon Rekognition Video 可以分析來自裝置攝影機的即時串流影片。如要從裝置來源存取媒體輸入，您需要安裝 GStreamer。GStreamer 是第三方多媒體框架軟件，將媒體源和處理工具在工作流程

管道中連接在一起。您還需要為 Gstreamer 安裝 Amazon Kinesis Video Streams 生成器外掛程式。此程序假設您已成功設定 Amazon Rekognition Video 和 Amazon Kinesis 資源。如需詳細資訊，請參閱設定您的 Amazon Rekognition Video 和 Amazon Kinesis 資源。

步驟 1：安裝 Gstreamer

下載並安裝第三方多媒體平台軟件 Gstreamer。您可以使用軟件包管理軟件，例如自製軟件 (自製軟件上的 Gstreamer)，也可以直接從免費桌面網站獲取。

通過從命令行終端啟動帶有測試源的影片源來驗證 Gstreamer 是否成功安裝。

```
$ gst-launch-1.0 videotestsrc ! autovideosink
```

步驟 2：安裝 Kinesis Video Streams 製作外掛程式

在本節中，您將下載 Amazon Kinesis Video Streams 生成器程式庫，並安裝 Kinesis Video Streams Gstreamer 外掛程式。

建立目錄，由 GitHub 儲存庫複製範例原始碼。請務必包括 `--recursive` 參數。

```
$ git clone --recursive https://github.com/awslabs/amazon-kinesis-video-streams-
producer-sdk-cpp.git
```

遵循資源庫提供的指示來規劃和建置專案。請務必針對您的作業系統使用平台的特定指令。當您執行 `cmake` 以安裝 Kinesis Video Streams Gstreamer 外掛程式時，請使用此 `-DBUILD_GSTREAMER_PLUGIN=ON` 參數。此專案需要安裝中包含的下列額外套件：GCC 或 Clang、Curl、Openssl 和 Log4cplus。如果您的組建因為遺失套件而失敗，請確認套件已安裝且已安裝在 PATH 中。如果在構建時遇到「無法執行 C 編譯程序」錯誤，請再次執行構建命令。有時，找不到正確的 C 編譯器。

執行下列命令，驗證 Kinesis Video Streams 外掛程式：

```
$ gst-inspect-1.0 kvssink
```

應該會顯示下列資訊，例如工廠和外掛程式詳細資訊：

```
Factory Details:
  Rank                     primary + 10 (266)
```

```
  Long-name                KVS Sink
  Klass                    Sink/Video/Network
  Description              GStreamer AWS KVS plugin
  Author                   AWS KVS <kinesis-video-support@amazon.com>

Plugin Details:
  Name                     kvssink
  Description              GStreamer AWS KVS plugin
  Filename                 /Users/YOUR_USER/amazon-kinesis-video-streams-producer-sdk-
cpp/build/libgstkvssink.so
  Version                  1.0
  License                  Proprietary
  Source module            kvssinkpackage
  Binary package           GStreamer
  Origin URL               http://gstreamer.net/

  ...
```

步驟 3：使用室壁執行影片流插件

在您開始從裝置攝影機串流到 Kinesis Video Streams 之前，您可能需要將媒體來源轉換為 Kinesis Video Streams 可接受的轉碼器。若要判斷目前連線至電腦之裝置的規格和格式功能，請執行下列命令。

```
$ gst-device-monitor-1.0
```

若要開始串流，請使用下列範例命令啟動 Gstreamer，然後新增您的憑證和 Amazon Kinesis Video Streams 資訊。應該針對您建立的 IAM 服務角色使用存取金鑰和區域，同時授予 Amazon Rekognition 存取 Kinesis 串流的權限。如需有關存取金鑰的詳細資訊，請參閱《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。另外，您可以根據使用情況的要求調整視訊格式參數，並可從裝置上獲得。

```
$ gst-launch-1.0 autovideosrc device=/dev/video0 ! videoconvert ! video/x-
raw,format=I420,width=640,height=480,framerate=30/1 !
                x264enc bframes=0 key-int-max=45 bitrate=500 ! video/x-h264,stream-
format=avc,alignment=au,profile=baseline !
                kvssink stream-name="YOUR_STREAM_NAME" storage-size=512 access-
key="YOUR_ACCESS_KEY" secret-key="YOUR_SECRET_ACCESS_KEY" aws-region="YOUR_AWS_REGION"
```

有關更多啟動命令，請參閱範例 GStreamer 啟動命令。

> **Note**
>
> 如果啟動指令因非交涉錯誤而終止，請檢查裝置監視器的輸出，並確定 `videoconvert` 參數值是裝置的有效功能。

幾秒鐘後，您將在 Kinesis 影片串流上看到來自裝置攝影機的影片摘要。若要開始使用 Amazon Rekognition 偵測和比對人臉，請啟動您的 Amazon Rekognition Video 串流處理器。如需詳細資訊，請參閱 亞馬遜視訊串流處理器操作概觀。

串流影片故障診斷

此主題提供使用 Amazon Rekognition Video 搭配串流影片的故障診斷資訊。

主題

- 我不知道我的串流處理器是否已成功建立
- 我不知道是否已正確設定我的串流處理器
- 我的串流處理器未傳回結果
- 我的串流處理器狀態是失敗
- 我的串流處理器未傳回預期的結果

我不知道我的串流處理器是否已成功建立

使用下面的 AWS CLI 命令來獲取流處理器及其當前狀態的列表。

```
aws rekognition list-stream-processors
```

您可以通過使用以下 AWS CLI 命令獲取其他詳細信息。將 `stream-processor-name` 取代為所需串流處理器的名稱。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

我不知道是否已正確設定我的串流處理器

如果您的程式碼不會輸出 Amazon Rekognition Video 的分析結果，則您的串流處理器可能未正確設定。執行以下專案確認您的串流處理器已正確設定並能夠產生結果。

若要判斷您的解決方案是否正確設定

1. 執行以下命令確認您的串流處理器處於執行狀態。將 stream-processor-name 變更為您的串流處理器名稱。若 Status 的值為 RUNNING，則串流處理器正在執行。若狀態為 RUNNING 而您未取得結果，請參閱我的串流處理器未傳回結果。若狀態為 FAILED，請參閱我的串流處理器狀態是失敗。

   ```
   aws rekognition describe-stream-processor --name stream-processor-name
   ```

2. 如果串流處理器正在執行，請執行下列 Bash 或 PowerShell 命令，從輸出 Kinesis 資料串流讀取資料。

   Bash

   ```
   SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000
    --shard-iterator-type TRIM_HORIZON --stream-name kinesis-data-stream-name --query
    'ShardIterator')
                           aws kinesis get-records --shard-iterator $SHARD_ITERATOR
   ```

   PowerShell

   ```
   aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-
   id shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name kinesis-
   data-stream-name).split('"')[4])
   ```

3. 使用 Base64 Decode 網站上的解碼工具將輸出解碼為人類可讀的字串。如需詳細資訊，請參閱步驟 3：取得記錄。
4. 如果命令可作用且您在 Kinesis 資料串流中看到臉部偵測結果，則您的解決方案已正確設定。如果命令失敗，請查看其他故障診斷建議並參閱 讓 Amazon Rekognition Video 有權存取您的資源。

或者，您也可以使用 "kinesis-process-record" AWS Lambda 藍圖將 Kinesis 資料串流中的訊息記錄到以 CloudWatch 進行連續視覺化。這會產生 AWS Lambda 和 CloudWatch的額外成本。

我的串流處理器未傳回結果

可能有幾個原因讓您的串流處理器無法傳回結果。

原因 1：您的串流處理器未正確設定

您的串流處理器可能未正確設定。如需詳細資訊，請參閱 我不知道是否已正確設定我的串流處理器。

原因 2：您的串流處理器未處於 RUNNING 狀態

故障診斷串流處理器的狀態

1. 使用以下 AWS CLI 命令檢查流處理器的狀態。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

2. 若 Status 的值為 STOPPED，請使用以下命令啟動串流處理器：

```
aws rekognition start-stream-processor --name stream-processor-name
```

3. 若 Status 的值為 FAILED，請參閱 我的串流處理器狀態是失敗。
4. 若 Status 的值為 STARTING，請等待 2 分鐘並重複步驟 1 以檢查狀態。如果狀態的值仍為 STARTING，請執行以下專案：

   a. 使用以下命令刪除串流處理器。

   ```
   aws rekognition delete-stream-processor --name stream-processor-name
   ```

   b. 使用相同組態建立新的串流處理器。如需詳細資訊，請參閱 使用串流視訊事件。

   c. 如果您仍然遇到問題，請聯絡 Sup AWS port 部門。
5. 若 Status 的值為 RUNNING，請參閱 原因 3：在 Kinesis 中沒有作用中的資料。

原因 3：在 Kinesis 中沒有作用中的資料

檢查 Kinesis 影片串流中是否有作用中的資料

1. 登入 AWS Management Console，然後開啟 Amazon Kinesis Video Streams 主控台，網址為 https://console.aws.amazon.com/kinesisvideo/。
2. 選取作為 Amazon Rekognition 串流處理器輸入的 Kinesis 影片串流。
3. 如果預覽顯示為無串流資料，則在輸入串流中沒有可供 Amazon Rekognition Video 處理的資料。

如需使用 Kinesis Video Streams 產生影片的資訊，請參閱 Kinesis 影片串流生成器程式庫。

我的串流處理器狀態是失敗

您可以通過使用以下 AWS CLI 命令檢查流處理器的狀態。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

如果狀態的值為失敗，請查看以下錯誤訊息的故障診斷資訊。

錯誤：「 角色存取遭拒」

由串流處理器使用的 IAM 角色不存在或 Amazon Rekognition Video 未具有擔任角色的許可。

故障診斷 IAM 角色的存取

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 在左側導覽窗格中，請選擇 Roles (角色) 並確認角色存在。
3. 如果角色存在，請檢查角色是否具有AmazonRekognitionServiceRole權限原則。
4. 若該角色不存在或未具有正確的許可，請參閱讓 Amazon Rekognition Video 有權存取您的資源。
5. 使用以下 AWS CLI 命令啟動流處理器。

```
aws rekognition start-stream-processor --name stream-processor-name
```

錯誤：「存取 Kinesis Video 遭拒或存取 Kinesis Data 遭拒。」

該角色無法存取 API 操作 `GetMedia` 和 `GetDataEndpoint`。該角色可能也無法存取 Kinesis Data Streams API 操作 `PutRecord` 和 `PutRecords`。

故障診斷 API 許可

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 開啟角色並確認角色已連接以下許可政策。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
```

```
            "Resource": "data-arn"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": "video-arn"
        }
    ]
}
```

3. 如果遺失任何許可，請更新政策。如需詳細資訊，請參閱 讓 Amazon Rekognition Video 有權存取您的資源。

錯誤：「流*input-video-stream-name*不存在」

Kinesis 對串流處理器的影片串流輸入不存在或設定不正確。

故障診斷 Kinesis 資料串流

1. 使用以下命令確認串流存在。

```
aws kinesisvideo list-streams
```

2. 如果串流存在，請檢查下列各項。

   - Amazon Resource Name (ARN) 與串流處理器的輸入串流 ARN 相同。
   - Kinesis 資料串流必須與串流處理器位於相同區域。

   如果串流處理器設定不正確，請使用下列 AWS CLI 指令將其刪除。

```
aws rekognition delete-stream-processor --name stream-processor-name
```

3. 以預期的 Kinesis 影片串流建立新的串流處理器。如需詳細資訊，請參閱 建立 Amazon Rekognition Video 人臉搜尋串流處理器。

錯誤：「找不到集合」

串流處理器用於比對人臉的 Amazon Rekognition 集合不存在或使用了錯誤的集合。

確認集合

1. 使用下面的 AWS CLI 命令來確定所需的集合是否存在。變更region為執行串流處理器的 AWS 地區。

   ```
   aws rekognition list-collections --region region
   ```

   如果所需集合不存在，請建立新的集合並新增臉部資訊。如需詳細資訊，請參閱 在集合中搜尋人臉。
2. 在您的呼叫中 CreateStreamProcessor，檢查CollectionId輸入參數的值是否正確。
3. 使用以下 AWS CLI 命令啟動流處理器。

   ```
   aws rekognition start-stream-processor --name stream-processor-name
   ```

錯誤：「未找到帳戶*## ID* 下的流*output-kinesis-data-stream##*」

串流處理器所使用的輸出 Kinesis 資料串流不存在於您的串流處理器， AWS 帳戶 或與串流處理器不在相同的 AWS 區域中。

Kinesis 資料串流的故障診斷

1. 使用下列 AWS CLI 命令來判斷 Kinesis 資料串流是否存在。變更region為您使用串流處理器的 AWS 地區。

   ```
   aws kinesis list-streams --region region
   ```
2. 若 Kinesis 資料串流存在，請確認 Kinesis 資料串流名稱與串流處理器使用的輸出串流名稱相同。
3. 如果 Kinesis 資料串流不存在，則可能存在於另一個 AWS 區域中。Kinesis 資料串流必須與串流處理器位於相同區域。
4. 如有必要，請建立新的 Kinesis 資料串流。

   a. 使用與串流處理器所用名稱相同的名稱建立 Kinesis 資料串流。如需詳細資訊，請參閱 步驟 1：建立資料串流。

   b. 使用以下 AWS CLI 命令啟動流處理器。

      ```
      aws rekognition start-stream-processor --name stream-processor-name
      ```

我的串流處理器未傳回預期的結果

如果您的串流處理器未傳回預期的臉部比對結果，請使用以下資訊。

- 在集合中搜尋人臉
- 相機設定的建議 (串流影片)

人物路徑

Amazon Rekognition Video 能建立人物在影片中的路徑資訊，例如：

- 偵測到路徑時，該人物出現在視訊影格中的位置。
- 偵測到的臉部特徵點，例如左眼的位置。

已儲存影片中的 Amazon Rekognition Video 人物路徑是一種非同步操作。要啟動人們在視頻通話StartPersonTracking的路徑。Amazon Rekognition Video 向其發佈影片分析操作的物件偵測結果和完成狀態的 Amazon Simple Notification Service 主題。如果視頻分析成功，請致電GetPersonTracking以獲取視頻分析的結果。如需呼叫 Amazon Rekognition Video API 操作的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

下列程序說明如何在儲存於 Amazon S3 儲存貯體的影片中追蹤人物路徑。此範例會展開 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中的程式碼，這會使用 Amazon Simple Queue Service 佇列來取得視訊分析要求的完成狀態。

若要偵測存放在 Amazon S3 儲存貯體 (SDK) 之視訊中的人物

1. 執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。
2. 將下列程式碼新增至您在步驟 1 中建立的類別 `VideoDetect`。

   Java

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

       //
Persons========================================================================
       private static void StartPersonDetection(String bucket, String video)
 throws Exception{


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

        StartPersonTrackingRequest req = new StartPersonTrackingRequest()
```

```
                .withVideo(new Video()
                        .withS3Object(new S3Object()
                            .withBucket(bucket)
                            .withName(video)))
                .withNotificationChannel(channel);



        StartPersonTrackingResult startPersonDetectionResult =
rek.startPersonTracking(req);
        startJobId=startPersonDetectionResult.getJobId();

       }

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

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

               personTrackingResult = rek.getPersonTracking(new
GetPersonTrackingRequest()
                    .withJobId(startJobId)
                    .withNextToken(paginationToken)
                    .withSortBy(PersonTrackingSortBy.TIMESTAMP)
                    .withMaxResults(maxResults));

               VideoMetadata
videoMetaData=personTrackingResult.getVideoMetadata();

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


               //Show persons, confidence and detection times
```

```
                List<PersonDetection> detectedPersons=
 personTrackingResult.getPersons();

                for (PersonDetection detectedPerson: detectedPersons) {

                   long seconds=detectedPerson.getTimestamp()/1000;
                   System.out.print("Sec: " + Long.toString(seconds) + " ");
                   System.out.println("Person Identifier: "  +
 detectedPerson.getPerson().getIndex());
                      System.out.println();
                }
            }  while (personTrackingResult !=null &&
 personTrackingResult.getNextToken() != null);

        }
```

在函數 main 中，將下行:

```
        StartLabelDetection(bucket, video);

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

取代為：

```
        StartPersonDetection(bucket, video);

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

Java V2

此代碼取自AWS文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.Video;
```

```
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.PersonDetection;
import java.util.List;

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

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

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

```
        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

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

        startPersonLabels(rekClient, channel, bucket, video);
        getPersonDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

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

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

            StartPersonTrackingRequest personTrackingRequest =
StartPersonTrackingRequest.builder()
                    .jobTag("DetectingLabels")
                    .video(vidOb)
                    .notificationChannel(channel)
                    .build();

            StartPersonTrackingResponse labelDetectionResponse =
rekClient.startPersonTracking(personTrackingRequest);
            startJobId = labelDetectionResponse.jobId();
```

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

    public static void getPersonDetectionResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetPersonTrackingResponse personTrackingResult = null;
            boolean finished = false;
            String status;
            int yy = 0;

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

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

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

                    personTrackingResult =
rekClient.getPersonTracking(recognitionRequest);
                    status = personTrackingResult.jobStatusAsString();

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

                finished = false;
```

```
                // Proceed when the job is done - otherwise VideoMetadata is
 null.
                VideoMetadata videoMetaData =
 personTrackingResult.videoMetadata();

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

                List<PersonDetection> detectedPersons =
 personTrackingResult.persons();
                for (PersonDetection detectedPerson : detectedPersons) {
                    long seconds = detectedPerson.timestamp() / 1000;
                    System.out.print("Sec: " + seconds + " ");
                    System.out.println("Person Identifier: " +
 detectedPerson.person().index());
                    System.out.println();
                }

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

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

Python

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

    # ============== People pathing ===============
    def StartPersonPathing(self):
        response=self.rek.start_person_tracking(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
```

```
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

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

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

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

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

            for personDetection in response['Persons']:
                print('Index: ' + str(personDetection['Person']['Index']))
                print('Timestamp: ' + str(personDetection['Timestamp']))
                print()

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

在函數 `main` 中，將下行:

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

取代為：

```
    analyzer.StartPersonPathing()
```

```
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetPersonPathingResults()
```

CLI

執行下列 AWS CLI 命令，在視訊中啟動人員路徑分析。

```
aws rekognition start-person-tracking --video "{"S3Object":{"Bucket":"bucket-
name","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-ARN","RoleArn":"role-ARN"}" \
--region region-name --profile profile-name
```

更新下列的值：

- 將 bucket-name 與 video-name 變更為您在步驟 2 中指定的 Amazon S3 儲存貯體與視訊檔名稱。
- 將 region-name 變更為您正在使用的 AWS 區域。
- 將建立 Rekognition 工作階段的行中 profile-name 的值取代為您的開發人員設定檔的名稱。
- 將 topic-ARN 變更為您在步驟 3 建立的 設定 Amazon Rekognition Video Amazon SNS 主題的 ARN。
- 將 role-ARN 變更為您在步驟 7 建立的 設定 Amazon Rekognition Video 服務角色的 ARN。

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。如需範例，請參閱下列內容：

```
aws rekognition start-person-tracking --video "{\"S3Object\":{\"Bucket\":
\"bucket-name\",\"Name\":\"video-name\"}}"
--notification-channel "{\"SNSTopicArn\":\"topic-ARN\",\"RoleArn\":\"role-ARN
\"}" \
--region region-name --profile profile-name
```

執行正在進行的程式碼範例之後，複製傳回的程式碼 jobID，並將其提供給下列 GetPersonTracking 命令，以 job-id-number 取代您之前收到的 jobID 結果：

```
aws rekognition get-person-tracking --job-id job-id-number
```

> **Note**
>
> 如果您已執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 以外的視訊範例，要取代的程式碼可能會不同。

3. 執行程式碼。將顯示追蹤到的人物之專有識別碼，同時以秒為單位顯示人物路徑被追蹤到的時間。

GetPersonTracking 作業回應

GetPersonTracking 傳回 Persons 物件的 PersonDetection 陣列，其中包含影片中追蹤到的人物詳細資訊，以及追蹤到他們路徑的時間。

您可以使用 SortBy 輸入參數對 Persons 排序。追蹤到影片中的人物路徑時，指定 TIMESTAMP 對元素排序。指定 INDEX 來排序影片中追蹤到的人物。在每個人物的結果組合中，依據追蹤路徑精確度的可信度遞減排序元素。在預設情況下，將根據 Persons 的排序傳回 TIMESTAMP。以下是來自 GetPersonDetection 的 JSON 回應範例。結果會依據人物路徑從影片開始後在視訊中被追蹤到的時間排序，以毫秒為單位。在回應中，請注意下列事項：

- 人物資訊：PersonDetection 陣列元素包含有關偵測到的人物之資訊。例如，該人物被偵測到的時間 (Timestamp)、該人物被偵測到時所在的視訊影格位置 (BoundingBox)、以及 Amazon Rekognition Video 對於正確偵測該人物的可信度 (Confidence)。

  不會在每次追蹤到人物路徑的時間戳記時傳回臉部特徵。此外在部分情況下，被追蹤到的人物身體可能無法顯示，因此只會傳回他們的臉部位置。
- 分頁資訊：範例顯示人物偵測資訊的一頁。您可以在 GetPersonTracking 的 MaxResults 輸入參數中指定要傳回的人物元素數目。如果結果數目超過 MaxResults，GetPersonTracking 會傳回用來取得下一頁結果的字符 (NextToken)。如需詳細資訊，請參閱 取得 Amazon Rekognition Video 分析結果。
- 索引：在影片中識別人物的專有識別碼。
- 視訊資訊：回應包含 GetPersonDetection 所傳回之每頁資訊中視訊格式 (VideoMetadata) 的相關資訊。

```
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "AcDymG0fSSoaI6+BBYpka5wVlqttysSPP8VvWcujMDluj1QpFo/vf
+mrMoqBGk8eUEiFlllR6g==",
    "Persons": [
        {
            "Person": {
                "BoundingBox": {
                    "Height": 0.8787037134170532,
                    "Left": 0.00572916679084301,
                    "Top": 0.12129629403352737,
                    "Width": 0.21666666865348816
                },
                "Face": {
                    "BoundingBox": {
                        "Height": 0.20000000298023224,
                        "Left": 0.029999999329447746,
                        "Top": 0.2199999988079071,
                        "Width": 0.11249999701976776
                    },
                    "Confidence": 99.85971069335938,
                    "Landmarks": [
                        {
                            "Type": "eyeLeft",
                            "X": 0.06842322647571564,
                            "Y": 0.3010137975215912
                        },
                        {
                            "Type": "eyeRight",
                            "X": 0.10543643683195114,
                            "Y": 0.29697132110595703
                        },
                        {
                            "Type": "nose",
                            "X": 0.09569807350635529,
                            "Y": 0.33701086044311523
                        },
                        {
                            "Type": "mouthLeft",
                            "X": 0.0732642263174057,
                            "Y": 0.3757539987564087
                        },
                        {
```

```
                        "Type": "mouthRight",
                        "X": 0.10589495301246643,
                        "Y": 0.3722417950630188
                    }
                ],
                "Pose": {
                    "Pitch": -0.5589138865470886,
                    "Roll": -5.1093974113464355,
                    "Yaw": 18.69594955444336
                },
                "Quality": {
                    "Brightness": 43.052337646484375,
                    "Sharpness": 99.68138885498047
                }
            },
            "Index": 0
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.9074074029922485,
                "Left": 0.24791666865348816,
                "Top": 0.09259258955717087,
                "Width": 0.375
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.23000000417232513,
                    "Left": 0.42500001192092896,
                    "Top": 0.16333332657814026,
                    "Width": 0.12937499582767487
                },
                "Confidence": 99.97504425048828,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.46415066719055176,
                        "Y": 0.2572723925113678
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.5068183541297913,
```

```
                        "Y": 0.23705792427062988
                    },
                    {
                        "Type": "nose",
                        "X": 0.49765899777412415,
                        "Y": 0.28383663296699524
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.487221896648407,
                        "Y": 0.3452930748462677
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.5142884850502014,
                        "Y": 0.33167609572410583
                    }
                ],
                "Pose": {
                    "Pitch": 15.966927528381348,
                    "Roll": -15.547388076782227,
                    "Yaw": 11.34195613861084
                },
                "Quality": {
                    "Brightness": 44.80223083496094,
                    "Sharpness": 99.95819854736328
                }
            },
            "Index": 1
        },
        "Timestamp": 0
    }.....

],
"VideoMetadata": {
    "Codec": "h264",
    "DurationMillis": 67301,
    "FileExtension": "mp4",
    "Format": "QuickTime / MOV",
    "FrameHeight": 1080,
    "FrameRate": 29.970029830932617,
    "FrameWidth": 1920
}
```

```
}
```

偵測個人防護裝備

Amazon Rekognition 可以偵測映像中人員佩戴的個人防護裝備 (PPE)。您可以使用此資訊來改善工作場所安全實踐。例如，您可以使用個人防護裝備偵測來協助判斷建築工地上的工人是否佩戴頭套，或醫務人員是否戴著面罩和手套。下圖顯示了一些可以偵測到的個人防護裝備類型。

要檢測圖像中的 PPE，請調用 DetectProtectiveEquipmentAPI 並傳遞輸入圖像。回應是包含下列專案的 JSON 結構。

- 映像中偵測到的人物。
- 佩戴個人防護裝備的身體部分 (面部、頭部、左手和右手)。
- 在身體部分 (面罩、手套和頭蓋) 上偵測到的個人防護裝備類型。
- 對於偵測到的個人防護裝備專案，指示個人防護裝備是否覆蓋相應的身體部分。

會針對映像中偵測到的個人和個人防護裝備的位置傳回邊界邊框。

您也可以選擇要求個人防護裝備專案摘要，以及在映像中偵測到的人員。如需詳細資訊，請參閱 摘要在映像中偵測到的個人防護裝備。

> **Note**
>
> Amazon Rekognition 個人防護裝備偵測不會執行人臉辨識或人臉比較，而且無法識別偵測到的人員。

個人防護裝備類型

DetectProtectiveEquipment檢測到以下類型的 PPE。如果您想要偵測映像中的其他類型的個人防護裝備，請考慮使用 Amazon Rekognition 自訂標籤來訓練自訂模型。如需詳細資訊，請參閱 Amazon Rekognition 自訂標籤。

面罩

DetectProtectiveEquipment 可以偵測常見的面罩，例如手術、N95 和布製的口罩。

手套

DetectProtectiveEquipment 可以偵測手套，如手術手套和安全手套。

頭蓋

DetectProtectiveEquipment 可以偵測安全帽和頭盔。

API 指出在映像中偵測到頭部、手部或面蓋。API 不傳回特定頭蓋類型的相關資訊。例如，手套類型的「手術手套」。

個人防護裝備偵測可信度

Amazon Rekognition 對映像中個人防護裝備、人員和身體部分的存在進行了預測。API 提供一個分數 (50-100)，指出 Amazon Rekognition 預測準確性的可信度。

> **Note**
>
> 如果您打算使用該 `DetectProtectiveEquipment` 操作做出影響個人權利、隱私權或服務存取權限的決定，我們建議您在採取行動之前將結果傳遞給人員進行審查和驗證。

摘要在映像中偵測到的個人防護裝備

您可以選擇要求個人防護裝備的專案摘要，以及在映像中偵測到的人員。您可以指定所需的防護裝備清單 (面罩、手套或頭蓋) 和最小受信度閾值 (例如 80%)。回應包括擁有所需個人防護裝備的人員、沒有所需個人防護裝備的人員以及無法作出判定的人員的整合每個映像識別碼 (ID) 摘要。

摘要可讓您快速回答問題，例如有多少人沒有佩戴面罩？還是每個人都使用個人防護裝備？ 摘要中偵測到的每個人都有一個唯一的 ID。您可以使用 ID 查找資訊，例如沒有使用個人防護裝備的人的邊界邊框位置。

> **Note**
>
> ID 是以每個映像分析為基礎隨機產生的，不同映像或對同一映像的多重分析不一致。

您可以概括面罩、頭套、手套或您選擇的組合。若要指定所需的個人防護裝備類型，請參閱 [指定彙總需求](#)。您也可以指定必須符合的最低受信度等級 (50-100)，才能將偵測納入摘要中。

如需摘要回應的詳細資訊 `DetectProtectiveEquipment`，請參閱 [了解回 DetectProtectiveEquipment 應](#)。

教學課程：建立使用 PPE 偵測影像的 AWS Lambda 函數

您可以在位於 Amazon S3 儲存貯體的映像中建立偵測個人防護設備 (PPE) 的 AWS Lambda 函數。有關此 Java V2 教程，請參閱[AWS 文檔 SDK 示例 GitHub 存儲庫](#)。

了解個人防護裝備偵測 API

下列資訊說明 [DetectProtectiveEquipment](#)API。如需範例程式碼，請參閱 [偵測映像中的個人防護裝備](#)。

提供映像

您可以將輸入映像 (JPG 或 PNG 格式) 作為映像位元組，或透過參考存放在 Amazon S3 儲存貯體中的映像來提供。

我們建議使用人臉朝向相機的映像。

如果您的輸入映像沒有旋轉至 0 度方向，建議您先將其旋轉至 0 度方向，然後再將其提交給 DetectProtectiveEquipment。JPG 格式的映像可能包含可交換映像檔案格式 (Exif) 中繼資料中的方向資訊。您可以使用此資訊來撰寫可輪換映像的程式碼。如需詳細資訊，請參閱可交換映像檔案格式版本 2.32。PNG 格式的映像不包含映像方向資訊。

要從 Amazon S3 存儲桶傳遞圖像，請使用至少具有 AmazonS3 ReadOnlyAccess 權限的用戶。使用擁有 AmazonRekognitionFullAccess 權限的使用者呼叫 DetectProtectiveEquipment.

在以下範例輸入 JSON 中，映像傳入 Amazon S3 儲存貯體中。如需詳細資訊，請參閱 使用映像。此範例要求所有個人防護裝備類型 (頭蓋、手套和面罩) 的摘要，偵測可信度 (MinConfidence) 為 80%。您應該指定介於 50% - 100% 之間的 MinConfidence 值，作為僅在偵測可信度介於 50% - 100% 之間的 DetectProtectiveEquipment 傳回的預測。如果您指定的值小於 50%，則結果會相同，指定 50% 的值。如需詳細資訊，請參閱 指定彙總需求。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "worker.jpg"
        }
    },
    "SummarizationAttributes": {
        "MinConfidence": 80,
        "RequiredEquipmentTypes": [
            "FACE_COVER",
            "HAND_COVER",
            "HEAD_COVER"
        ]
    }
}
```

如果您要處理大型映像集合，請考慮使用 AWS Batch 在背景批次處理 DetectProtectiveEquipment 的呼叫。

指定彙總需求

您可以選擇性地使用 SummarizationAttributes (ProtectiveEquipmentSummarizationAttributes) 輸入參數來要求影像中偵測到的 PPE 類型的摘要資訊。

若要指定要摘要的個人防護裝備類型，請使用 RequiredEquipmentTypes 陣列欄位。在陣列中，包含一或多個 FACE_COVER、HAND_COVER 或 HEAD_COVER。

使用此 MinConfidence 欄位可指定最小偵測可信度 (50 - 100)。摘要不包括人員、身體部分、身體部分覆蓋以及個人防護裝備物品，偵測可信度低於 MinConfidence。

如需 DetectProtectiveEquipment 回應格式的相關資訊，請參閱 了解回 DetectProtectiveEquipment 應。

了解回 DetectProtectiveEquipment 應

DetectProtectiveEquipment 傳回在輸入映像中偵測的一群人。對於每個人，傳回有關偵測到的身體部分和偵測到的個人防護裝備物品的資訊。戴著頭套、手套和面罩的工作人員的以下映像的 JSON 如下。

在 JSON 中，請注意下列事項。

- 偵測人員：Persons 是在映像上偵測到的一系列人員 (包括未佩戴個人防護裝備的人員)。DetectProtectiveEquipment 可以在映像中偵測到的最多 15 個人身上偵測到個人防護裝備。陣列中的每個ProtectiveEquipmentPerson物件都包含人員 ID、人員的邊界框、偵測到的身體部位，以及偵測到的 PPE 項目。ProtectiveEquipmentPerson 中的 Confidence 值表示 Amazon Rekognition 對邊界邊框包含人員的百分比可信度。
- 身體部位 — BodyParts 是在一個人身上檢測到的一系列身體部位 (包括 PPE 未涵蓋的身體部位) 。ProtectiveEquipmentBodyPart每個 ProtectiveEquipmentBodyPart 都包括偵測到的本體零件的名稱 (Name)。DetectProtectEquipment 可以偵測人臉、頭部、左側和右側身體部分。ProtectiveEquipmentBodyPart 中的 Confidence 欄位表示 Amazon Rekognition 對身體部分偵測準確度的百分比可信度。
- 個人防護裝備專案：ProtectiveEquipmentBodyPart 物件內的陣列 EquipmentDetections 包含偵測到的多个個人防護裝備專案。每個EquipmentDetection物件都包含下列欄位。

- Type：偵測到的個人防護裝備類型。
- BoundingBox：圍繞偵測到的個人防護裝備的邊界邊框。
- Confidence：Amazon Rekognition 邊界邊框包含偵測到的個人防護裝備的受信度。
- CoversBodyPart：指示偵測到的個人防護裝備是否在相應的身體部分。

此CoversBodyPart欄位Value為布林值，指出偵測到的 PPE 是否在對應的主體部位上。該欄位 Confidence 表明預測的可信度。您可以使用 CoversBodyPart 過濾掉偵測到的個人防護裝備在映像中但實際上不是在個人身上的案例。

> **Note**
>
> CoversBodyPart 表明且該人沒有受到防護裝備的充分保護，或者未正確佩戴防護裝備。

- 摘要資訊：Summary 包含 SummarizationAttributes 輸入參數中指定的摘要資訊。如需詳細資訊，請參閱 指定彙總需求。

  Summary是一個包含以下信息ProtectiveEquipmentSummary的類型的對象。

  - PersonsWithRequiredEquipment：符合以下條件的每個人的 ID。
    - 該人員穿著 SummarizationAttributes 輸入參數中指定的所有個人防護裝備。
    - 用於人員 (ProtectiveEquipmentPerson)、身體部分 (ProtectiveEquipmentBodyPart)、防護裝備 (EquipmentDetection)的 Confidence 等於或大於指定的最小置信閾值 (MinConfidence)。
    - 個人防護裝備的所有項目的 CoversBodyPart 值為真。
  - PersonsWithoutRequiredEquipment：符合下列條件之一的人員的 ID。
    - 人員(ProtectiveEquipmentPerson)、身體部分 (ProtectiveEquipmentBodyPart) 和身體部分覆蓋率 (CoversBodyPart)的 Confidence 值大於指定的最小受信度閾值 (MinConfidence)，但該人員缺少一個或多個指定的個人防護裝備 (SummarizationAttributes)。
    - 如果用于任何指定個人防護裝備 (SummarizationAttributes) 的 Confidence 值大於指定的最小可信度閾值 (MinConfidence)，則 CoversBodyPart 值為假。此人還具有所有指定的個人防護裝備 (SummarizationAttributes)、且針對人員 (ProtectiveEquipmentPerson)、身體部分 (ProtectiveEquipmentBodyPart) 和防護裝置 (EquipmentDetection) 的 Confidence 值大於或等於最小可信度閾值 (MinConfidence。

- PersonsIndeterminate：人員 (ProtectiveEquipmentPerson)、身體部分 (ProtectiveEquipmentBodyPart)、防護裝備 (EquipmentDetection) 的 Confidence 值所在位置偵測到的人員 ID 陣列或 CoversBodyPart 布林值低於指定的最小受信度閾值 (MinConfidence)。

使用數組大小來獲取特定摘要的計數。例如，PersonsWithRequiredEquipment 值大小表明偵測到佩戴指定類型個人防護裝備的人數。

您可以使用人員 ID 來尋找有關人員的進一步資訊，例如人物的邊界邊框位置。人員 ID 映射至 Persons (ProtectiveEquipmentPerson 的陣列) 中傳回的 ProtectiveEquipmentPerson 物件 ID 欄位。然後，您可以從相應的 ProtectiveEquipmentPerson 物件中獲取邊界邊框和其他資訊。


```
{
   "ProtectiveEquipmentModelVersion": "1.0",
   "Persons": [
       {
           "BodyParts": [
               {
                   "Name": "FACE",
                   "Confidence": 99.99861145019531,
                   "EquipmentDetections": [
                       {
                           "BoundingBox": {
                               "Width": 0.14528800547122955,
                               "Height": 0.14956723153591156,
                               "Left": 0.4363413453102112,
                               "Top": 0.34203192591667175
                           },
                           "Confidence": 99.90001678466797,
                           "Type": "FACE_COVER",
                           "CoversBodyPart": {
                               "Confidence": 98.0676498413086,
                               "Value": true
                           }
                       }
                   ]
               },
               {
                   "Name": "LEFT_HAND",
```

```
                    "Confidence": 96.9786376953125,
                    "EquipmentDetections": [
                        {
                            "BoundingBox": {
                                "Width": 0.14495663344860077,
                                "Height": 0.12936046719551086,
                                "Left": 0.5114737153053284,
                                "Top": 0.5744519829750061
                            },
                            "Confidence": 83.72270965576172,
                            "Type": "HAND_COVER",
                            "CoversBodyPart": {
                                "Confidence": 96.9288558959961,
                                "Value": true
                            }
                        }
                    ]
                },
                {
                    "Name": "RIGHT_HAND",
                    "Confidence": 99.82939147949219,
                    "EquipmentDetections": [
                        {
                            "BoundingBox": {
                                "Width": 0.20971858501434326,
                                "Height": 0.20528452098369598,
                                "Left": 0.2711356580257416,
                                "Top": 0.6750612258911133
                            },
                            "Confidence": 95.70789337158203,
                            "Type": "HAND_COVER",
                            "CoversBodyPart": {
                                "Confidence": 99.85433197021484,
                                "Value": true
                            }
                        }
                    ]
                },
                {
                    "Name": "HEAD",
                    "Confidence": 99.9999008178711,
                    "EquipmentDetections": [
                        {
                            "BoundingBox": {
```

```
                                "Width": 0.24350935220718384,
                                "Height": 0.34623199701309204,
                                "Left": 0.43011072278022766,
                                "Top": 0.01103297434747219
                            },
                            "Confidence": 83.88762664794922,
                            "Type": "HEAD_COVER",
                            "CoversBodyPart": {
                                "Confidence": 99.96485900878906,
                                "Value": true
                            }
                        }
                    ]
                }
            ],
            "BoundingBox": {
                "Width": 0.7403100728988647,
                "Height": 0.9412225484848022,
                "Left": 0.02214839495718479,
                "Top": 0.03134796395897865
            },
            "Confidence": 99.98855590820312,
            "Id": 0
        }
    ],
    "Summary": {
        "PersonsWithRequiredEquipment": [
            0
        ],
        "PersonsWithoutRequiredEquipment": [],
        "PersonsIndeterminate": []
    }
}
```


偵測映像中的個人防護裝備

若要偵測影像中人員的個人防護裝備 (PPE)，請使用DetectProtectiveEquipment非儲存 API 作業。

您可以使用 AWS SDK 或 AWS Command Line Interface (AWS CLI)，以映像位元組陣列 (Base64 編碼映像位元組) 或 Amazon S3 物件的方式提供輸入映像。這些範例使用存放在 Amazon S3 儲存貯體中的映像。如需詳細資訊，請參閱 使用映像。

若要偵測映像中人物身上的個人防護裝備

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 將 (含有一或多個配備個人防護裝備的人員) 的映像上傳至您的 S3 儲存貯體。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例來呼叫 DetectProtectiveEquipment 操作。如需有關在映像中顯示邊界邊框的資訊，請參閱 顯示週框方塊。

   Java

   此範例顯示在映像中偵測到的人員身上偵測到的個人防護裝備專案的相關資訊。

   將 bucket 的值變更為包含映像檔案的 Amazon S3 儲存貯體名稱。變更映像檔案名稱的 photo 值。

```
//Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.samples;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentBodyPart;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentPerson;
import
 com.amazonaws.services.rekognition.model.ProtectiveEquipmentSummarizationAttributes;

import java.util.List;
import com.amazonaws.services.rekognition.model.BoundingBox;
```

```
import
 com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentRequest;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentResult;
import com.amazonaws.services.rekognition.model.EquipmentDetection;


public class DetectPPE {

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

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


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        ProtectiveEquipmentSummarizationAttributes summaryAttributes = new
 ProtectiveEquipmentSummarizationAttributes()
                .withMinConfidence(80F)
                .withRequiredEquipmentTypes("FACE_COVER", "HAND_COVER",
 "HEAD_COVER");

        DetectProtectiveEquipmentRequest request = new
 DetectProtectiveEquipmentRequest()
                .withImage(new Image()
                        .withS3Object(new S3Object()
                                .withName(photo).withBucket(bucket)))
                .withSummarizationAttributes(summaryAttributes);

        try {
            System.out.println("Detected PPE for people in image " + photo);
            System.out.println("Detected people\n---------------");
            DetectProtectiveEquipmentResult result =
 rekognitionClient.detectProtectiveEquipment(request);


            List <ProtectiveEquipmentPerson> persons = result.getPersons();


            for (ProtectiveEquipmentPerson person: persons) {
                System.out.println("ID: " + person.getId());
                List<ProtectiveEquipmentBodyPart>
 bodyParts=person.getBodyParts();
```

```
                if (bodyParts.isEmpty()){
                    System.out.println("\tNo body parts detected");
                } else
                    for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                        System.out.println("\t" + bodyPart.getName() + ".
Confidence: " + bodyPart.getConfidence().toString());



                        List<EquipmentDetection>
equipmentDetections=bodyPart.getEquipmentDetections();

                        if (equipmentDetections.isEmpty()){
                            System.out.println("\t\tNo PPE Detected on " +
bodyPart.getName());

                        }
                        else {
                            for (EquipmentDetection item: equipmentDetections) {
                                System.out.println("\t\tItem: " + item.getType()
+ ". Confidence: " + item.getConfidence().toString());
                                System.out.println("\t\tCovers body part: "
                                        +
item.getCoversBodyPart().getValue().toString() + ". Confidence: " +
item.getCoversBodyPart().getConfidence().toString());

                                System.out.println("\t\tBounding Box");
                                BoundingBox box =item.getBoundingBox();

                                System.out.println("\t\tLeft: "
+box.getLeft().toString());
                                System.out.println("\t\tTop: " +
box.getTop().toString());
                                System.out.println("\t\tWidth: " +
box.getWidth().toString());
                                System.out.println("\t\tHeight: " +
box.getHeight().toString());
                                System.out.println("\t\tConfidence: " +
item.getConfidence().toString());
                                System.out.println();
                            }
                        }

                    }
```

```
            }
            System.out.println("Person ID Summary\n-----------------");

            //List<Integer> list=;
            DisplaySummary("With required equipment",
 result.getSummary().getPersonsWithRequiredEquipment());
            DisplaySummary("Without required equipment",
 result.getSummary().getPersonsWithoutRequiredEquipment());
            DisplaySummary("Indeterminate",
 result.getSummary().getPersonsIndeterminate());


        } catch(AmazonRekognitionException e) {
            e.printStackTrace();
        }
    }
    static void DisplaySummary(String summaryType,List<Integer> idList)
    {
        System.out.print(summaryType + "\n\tIDs  ");
        if (idList.size()==0) {
            System.out.println("None");
        }
        else {
            int count=0;
            for (Integer id: idList ) {
                if (count++ == idList.size()-1) {
                    System.out.println(id.toString());
                }
                else {
                    System.out.print(id.toString() + ", ");
                }
            }
        }

        System.out.println();


    }
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.detect_ppe.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_ppe.import]

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

 public static void main(String[] args) {

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

```
        "Usage: " +
        "   <sourceImage> <bucketName>\n\n" +
        "Where:\n" +
        "   sourceImage - The name of the image in an Amazon S3 bucket (for
example, people.png). \n\n" +
        "   bucketName - The name of the Amazon S3 bucket (for example,
myBucket). \n\n";

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

    String sourceImage = args[0];
    String bucketName = args[1];
    Region region = Region.US_WEST_2;
    S3Client s3 = S3Client.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

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

    displayGear(s3, rekClient, sourceImage, bucketName) ;
    s3.close();
    rekClient.close();
    System.out.println("This example is done!");
}

// snippet-start:[rekognition.java2.detect_ppe.main]
public static void displayGear(S3Client s3,
                               RekognitionClient rekClient,
                               String sourceImage,
                               String bucketName) {

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

    try {
        ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
ProtectiveEquipmentSummarizationAttributes.builder()
```

```
            .minConfidence(80F)
            .requiredEquipmentTypesWithStrings("FACE_COVER", "HAND_COVER",
"HEAD_COVER")
            .build();

        SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
        software.amazon.awssdk.services.rekognition.model.Image souImage =
Image.builder()
            .bytes(sourceBytes)
            .build();

        DetectProtectiveEquipmentRequest request =
DetectProtectiveEquipmentRequest.builder()
            .image(souImage)
            .summarizationAttributes(summarizationAttributes)
            .build();

        DetectProtectiveEquipmentResponse result =
rekClient.detectProtectiveEquipment(request);
        List<ProtectiveEquipmentPerson> persons = result.persons();
        for (ProtectiveEquipmentPerson person: persons) {
            System.out.println("ID: " + person.id());
            List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
            if (bodyParts.isEmpty()){
                System.out.println("\tNo body parts detected");
            } else
                for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                    System.out.println("\t" + bodyPart.name() + ". Confidence:
" + bodyPart.confidence().toString());
                    List<EquipmentDetection>
equipmentDetections=bodyPart.equipmentDetections();

                    if (equipmentDetections.isEmpty()){
                        System.out.println("\t\tNo PPE Detected on " +
bodyPart.name());
                    } else {
                        for (EquipmentDetection item: equipmentDetections) {
                            System.out.println("\t\tItem: " + item.type() + ".
Confidence: " + item.confidence().toString());
                            System.out.println("\t\tCovers body part: "
                                    + item.coversBodyPart().value().toString()
+ ". Confidence: " + item.coversBodyPart().confidence().toString());

                            System.out.println("\t\tBounding Box");
```

```
                            BoundingBox box =item.boundingBox();
                            System.out.println("\t\tLeft: "
+box.left().toString());
                            System.out.println("\t\tTop: " +
box.top().toString());
                            System.out.println("\t\tWidth: " +
box.width().toString());
                            System.out.println("\t\tHeight: " +
box.height().toString());
                            System.out.println("\t\tConfidence: " +
item.confidence().toString());
                            System.out.println();
                        }
                    }
                }
        }
        System.out.println("Person ID Summary\n-----------------");

        displaySummary("With required equipment",
result.summary().personsWithRequiredEquipment());
        displaySummary("Without required equipment",
result.summary().personsWithoutRequiredEquipment());
        displaySummary("Indeterminate",
result.summary().personsIndeterminate());

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

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

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

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

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

    static void displaySummary(String summaryType,List<Integer> idList) {
        System.out.print(summaryType + "\n\tIDs  ");
        if (idList.size()==0) {
            System.out.println("None");
        } else {
            int count=0;
            for (Integer id: idList ) {
                if (count++ == idList.size()-1) {
                    System.out.println(id.toString());
                } else {
                    System.out.print(id.toString() + ", ");
                }
            }
        }
        System.out.println();
    }
    // snippet-end:[rekognition.java2.detect_ppe.main]
}
```

AWS CLI

此 AWS CLI 命令要求 PPE 摘要，並顯示 `detect-protective-equipment` CLI 作業的 JSON 輸出。

將 `bucketname` 變更為 Amazon S3 儲存貯體的名稱，其中包含映像。將 `input.jpg` 變更為您要使用的映像名稱。

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=bucketname,Name=input.jpg}" \
  --summarization-attributes
 "MinConfidence=80,RequiredEquipmentTypes=['FACE_COVER','HAND_COVER','HEAD_COVER']"
```

此 AWS CLI 命令會顯示 `detect-protective-equipment` CLI 作業的 JSON 輸出。

將 bucketname 變更為 Amazon S3 儲存貯體的名稱，其中包含映像。將 input.jpg 變更為您要使用的映像名稱。

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

Python

此範例顯示在映像中偵測到的人員身上偵測到的個人防護裝備專案的相關資訊。

將 bucket 的值變更為包含映像檔案的 Amazon S3 儲存貯體名稱。變更映像檔案名稱的 photo 值。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
# Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def detect_ppe(photo, bucket):

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

    response = client.detect_protective_equipment(Image={'S3Object': {'Bucket':
 bucket, 'Name': photo}},

 SummarizationAttributes={'MinConfidence': 80,

 'RequiredEquipmentTypes': ['FACE_COVER',

                     'HAND_COVER',

                     'HEAD_COVER']})

    print('Detected PPE for people in image ' + photo)
    print('\nDetected people\n---------------')
    for person in response['Persons']:

        print('Person ID: ' + str(person['Id']))
```

```
        print('Body Parts\n----------')
        body_parts = person['BodyParts']
        if len(body_parts) == 0:
            print('No body parts found')
        else:
            for body_part in body_parts:
                print('\t' + body_part['Name'] + '\n\t\tConfidence: ' +
 str(body_part['Confidence']))
                print('\n\t\tDetected PPE\n\t\t------------')
                ppe_items = body_part['EquipmentDetections']
                if len(ppe_items) == 0:
                    print('\t\tNo PPE detected on ' + body_part['Name'])
                else:
                    for ppe_item in ppe_items:
                        print('\t\t' + ppe_item['Type'] + '\n\t\t\tConfidence: '
 + str(ppe_item['Confidence']))
                        print('\t\tCovers body part: ' + str(
                            ppe_item['CoversBodyPart']['Value']) + '\n\t\t
\tConfidence: ' + str(
                            ppe_item['CoversBodyPart']['Confidence']))
                        print('\t\tBounding Box:')
                        print('\t\t\tTop: ' + str(ppe_item['BoundingBox']
['Top']))
                        print('\t\t\tLeft: ' + str(ppe_item['BoundingBox']
['Left']))
                        print('\t\t\tWidth: ' + str(ppe_item['BoundingBox']
['Width']))
                        print('\t\t\tHeight: ' + str(ppe_item['BoundingBox']
['Height']))
                        print('\t\t\tConfidence: ' +
 str(ppe_item['Confidence']))
            print()
        print()

    print('Person ID Summary\n----------------')
    display_summary('With required equipment', response['Summary']
['PersonsWithRequiredEquipment'])
    display_summary('Without required equipment', response['Summary']
['PersonsWithoutRequiredEquipment'])
    display_summary('Indeterminate', response['Summary']
['PersonsIndeterminate'])

    print()
    return len(response['Persons'])
```

```
# Display summary information for supplied summary.
def display_summary(summary_type, summary):
    print(summary_type + '\n\tIDs: ', end='')
    if (len(summary) == 0):
        print('None')
    else:
        for num, id in enumerate(summary, start=0):
            if num == len(summary) - 1:
                print(id)
            else:
                print(str(id) + ', ', end='')

def main():
    photo = 'photo-name'
    bucket = 'bucket-name'
    person_count = detect_ppe(photo, bucket)
    print("Persons detected: " + str(person_count))

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

範例：繪製面蓋周圍的邊界邊框

下列範例說明如何在偵測到的人臉蓋周圍繪製邊界邊框。如需使用 AWS Lambda 和 Amazon DynamoDB 的範例，請參閱AWS 文件開發套件範例 GitHub 儲存庫。

要檢測面罩，請使用DetectProtectiveEquipment非存儲 API 操作。映像會從本機檔案系統載入。您用映像位元組陣列 (Base64 編碼映像位元組) 的方式提供 `DetectProtectiveEquipment` 的輸入映像。如需詳細資訊，請參閱 使用映像。

此範例會在偵測到的面蓋周圍顯示邊界邊框。如果面蓋完全覆蓋身體部分，則邊界邊框為綠色。否則會顯示紅色的邊界邊框。作為警告，如果偵測可信度低於指定的受信度值，則面蓋邊界邊框內會顯示黃色邊界邊框。如果未偵測到面蓋，則會在人物周圍繪製紅色邊框。

影像輸出類似於以下內容。

在偵測到的面蓋上顯示邊界邊框

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 許可的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用下列範例來呼叫 DetectProtectiveEquipment 操作。如需有關在映像中顯示邊界邊框的資訊，請參閱 顯示週框方塊。

   Java

   在函數 main 中，變更下列專案：

   - 本機映像檔案 (PNG 或 JPEG) 的 photo 路徑和檔案名稱的值。

- confidence 的值達到所需的可信度等級 (50-100)。

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


import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import com.amazonaws.util.IOUtils;

import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.BoundingBox;
import
 com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentRequest;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentResult;
import com.amazonaws.services.rekognition.model.EquipmentDetection;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentBodyPart;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentPerson;

// Calls DetectFaces and displays a bounding box around each detected image.
public class PPEBoundingBox extends JPanel {

    private static final long serialVersionUID = 1L;

    BufferedImage image;
    static int scale;
    DetectProtectiveEquipmentResult result;
```

```
    float confidence=80;

    public PPEBoundingBox(DetectProtectiveEquipmentResult ppeResult,
BufferedImage bufImage, float requiredConfidence) throws Exception {
        super();
        scale = 2; // increase to shrink image size.

        result = ppeResult;
        image = bufImage;

        confidence=requiredConfidence;
    }
    // Draws the bounding box around the detected faces.
    public void paintComponent(Graphics g) {
        float left = 0;
        float top = 0;
        int height = image.getHeight(this);
        int width = image.getWidth(this);
        int offset=20;

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

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

        // Iterate through detected persons and display bounding boxes.
        List<ProtectiveEquipmentPerson> persons = result.getPersons();

        for (ProtectiveEquipmentPerson person: persons) {
            BoundingBox boxPerson = person.getBoundingBox();
            left = width * boxPerson.getLeft();
            top = height * boxPerson.getTop();
            Boolean foundMask=false;

            List<ProtectiveEquipmentBodyPart> bodyParts=person.getBodyParts();

            if (bodyParts.isEmpty()==false)
             {
                    //body parts detected

                    for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
```

```
                        List<EquipmentDetection>
equipmentDetections=bodyPart.getEquipmentDetections();

                        for (EquipmentDetection item: equipmentDetections) {

                            if (item.getType().contentEquals("FACE_COVER"))
                            {
                                // Draw green or red bounding box depending on
mask coverage.
                                foundMask=true;
                                BoundingBox box =item.getBoundingBox();
                                left = width * box.getLeft();
                                top = height * box.getTop();
                                Color maskColor=new Color( 0, 212, 0);

                                if (item.getCoversBodyPart().getValue()==false)
{
                                    // red bounding box
                                    maskColor=new Color( 255, 0, 0);
                                }
                                g2d.setColor(maskColor);
                                g2d.drawRect(Math.round(left / scale),
Math.round(top / scale),
                                        Math.round((width * box.getWidth()) /
scale), Math.round((height * box.getHeight())) / scale);

                                // Check confidence is > supplied confidence.
                                if (item.getCoversBodyPart().getConfidence()<
confidence)
                                {
                                    // Draw a yellow bounding box inside face
mask bounding box
                                    maskColor=new Color( 255, 255, 0);
                                    g2d.setColor(maskColor);
                                    g2d.drawRect(Math.round((left + offset) /
scale),
                                            Math.round((top + offset) / scale),
                                            Math.round((width *
box.getWidth())- (offset * 2 ))/ scale,
                                            Math.round((height *
box.getHeight()) -( offset* 2)) / scale);
                                }

                            }
```

```
                        }

                    }

                }

            // Didn't find a mask, so draw person bounding box red
            if (foundMask==false) {

                left = width * boxPerson.getLeft();
                top = height * boxPerson.getTop();
                g2d.setColor(new Color(255, 0, 0));
                g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                        Math.round(((width) * boxPerson.getWidth()) / scale),
Math.round((height * boxPerson.getHeight())) / scale);
            }
        }

   }


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

       String photo = "photo";

       float confidence =80;


       int height = 0;
       int width = 0;

       BufferedImage image = null;
       ByteBuffer imageBytes;

       // Get image bytes for call to DetectProtectiveEquipment
       try (InputStream inputStream = new FileInputStream(new File(photo))) {
           imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
       }

       //Get image for display
       InputStream imageBytesStream;
       imageBytesStream = new ByteArrayInputStream(imageBytes.array());

       ByteArrayOutputStream baos = new ByteArrayOutputStream();
```

```
        image=ImageIO.read(imageBytesStream);
        ImageIO.write(image, "jpg", baos);
        width = image.getWidth();
        height = image.getHeight();

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


        // Call DetectProtectiveEquipment
        DetectProtectiveEquipmentRequest request = new
 DetectProtectiveEquipmentRequest()
                .withImage(new Image()
                        .withBytes(imageBytes));

        DetectProtectiveEquipmentResult result =
 rekognitionClient.detectProtectiveEquipment(request);


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

    }
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.List;
import javax.imageio.ImageIO;
```

```
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
//snippet-end:[rekognition.java2.display_mask.import]

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

 DetectProtectiveEquipmentResponse result;
 static BufferedImage image;
 static int scale;
 float confidence;

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

```
    final String usage = "\n" +
        "Usage: " +
        "   <sourceImage> <bucketName>\n\n" +
        "Where:\n" +
        "   sourceImage - The name of the image in an Amazon S3 bucket that
 shows a person wearing a mask (for example, masks.png). \n\n" +
        "   bucketName - The name of the Amazon S3 bucket (for example,
 myBucket). \n\n";

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

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

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

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

 // snippet-start:[rekognition.java2.display_mask.main]
public static void displayGear(S3Client s3,
                                   RekognitionClient rekClient,
                                   String sourceImage,
                                   String bucketName) {
   float confidence = 80;
   byte[] data = getObjectBytes(s3, bucketName, sourceImage);
   InputStream is = new ByteArrayInputStream(data);

   try {
       ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
 ProtectiveEquipmentSummarizationAttributes.builder()
```

```
            .minConfidence(70F)
            .requiredEquipmentTypesWithStrings("FACE_COVER")
            .build();

        SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
        image = ImageIO.read(sourceBytes.asInputStream());

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

        DetectProtectiveEquipmentRequest request =
 DetectProtectiveEquipmentRequest.builder()
            .image(souImage)
            .summarizationAttributes(summarizationAttributes)
            .build();

        DetectProtectiveEquipmentResponse result =
 rekClient.detectProtectiveEquipment(request);
        JFrame frame = new JFrame("Detect PPE");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        PPEBoundingBoxFrame panel = new PPEBoundingBoxFrame(result, image,
 confidence);
        panel.setPreferredSize(new Dimension(image.getWidth() / scale,
 image.getHeight() / scale));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    } catch (RekognitionException e) {
        e.printStackTrace();
        System.exit(1);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

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

     try {
         GetObjectRequest objectRequest = GetObjectRequest
```

```
            .builder()
            .key(keyName)
            .bucket(bucketName)
            .build();

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

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

public PPEBoundingBoxFrame(DetectProtectiveEquipmentResponse ppeResult,
BufferedImage bufImage, float requiredConfidence) {
    super();
    scale = 1; // increase to shrink image size.
    result = ppeResult;
    image = bufImage;
    confidence=requiredConfidence;
}

// Draws the bounding box around the detected masks.
public void paintComponent(Graphics g) {
    float left = 0;
    float top = 0;
    int height = image.getHeight(this);
    int width = image.getWidth(this);
    int offset=20;

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

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

    // Iterate through detected persons and display bounding boxes.
    List<ProtectiveEquipmentPerson> persons = result.persons();
    for (ProtectiveEquipmentPerson person: persons) {

        List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
```

```
        if (!bodyParts.isEmpty()){
            for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                List<EquipmentDetection>
 equipmentDetections=bodyPart.equipmentDetections();
                for (EquipmentDetection item: equipmentDetections) {

                    String myType = item.type().toString();
                    if (myType.compareTo("FACE_COVER") ==0) {

                        // Draw green bounding box depending on mask coverage.
                        BoundingBox box =item.boundingBox();
                        left = width * box.left();
                        top = height * box.top();
                        Color maskColor=new Color( 0, 212, 0);

                        if (item.coversBodyPart().equals(false)) {
                            // red bounding box.
                            maskColor=new Color( 255, 0, 0);
                        }
                        g2d.setColor(maskColor);
                        g2d.drawRect(Math.round(left / scale), Math.round(top /
 scale),
                                Math.round((width * box.width()) / scale),
 Math.round((height * box.height())) / scale);

                        // Check confidence is > supplied confidence.
                        if (item.coversBodyPart().confidence() < confidence) {
                            // Draw a yellow bounding box inside face mask
 bounding box.
                            maskColor=new Color( 255, 255, 0);
                            g2d.setColor(maskColor);
                            g2d.drawRect(Math.round((left + offset) / scale),
                                    Math.round((top + offset) / scale),
                                    Math.round((width * box.width())- (offset *
 2 ))/ scale,
                                    Math.round((height * box.height()) -
( offset* 2)) / scale);
                        }
                    }
                }
            }
        }
    }
 }
```

```
 // snippet-end:[rekognition.java2.display_mask.main]
}
```

Python

在函數 main 中，變更下列專案：

- 本機映像檔案 (PNG 或 JPEG) 的 photo 路徑和檔案名稱的值。
- confidence 的值達到所需的可信度等級 (50-100)。
- 將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
#Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

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

def detect_ppe(photo, confidence):

    fill_green='#00d400'
    fill_red='#ff0000'
    fill_yellow='#ffff00'
    line_width=3

    #open image and get image data from stream.
    image = Image.open(open(photo,'rb'))
    stream = io.BytesIO()
    image.save(stream, format=image.format)
    image_binary = stream.getvalue()
    imgWidth, imgHeight = image.size
    draw = ImageDraw.Draw(image)

    client=boto3.client('rekognition')

    response = client.detect_protective_equipment(Image={'Bytes': image_binary})

    for person in response['Persons']:

        found_mask=False
```

```
        for body_part in person['BodyParts']:
            ppe_items = body_part['EquipmentDetections']

            for ppe_item in ppe_items:
                #found a mask
                if ppe_item['Type'] == 'FACE_COVER':
                    fill_color=fill_green
                    found_mask=True
                    # check if mask covers face
                    if ppe_item['CoversBodyPart']['Value'] == False:
                        fill_color=fill='#ff0000'
                    # draw bounding box around mask
                    box = ppe_item['BoundingBox']
                    left = imgWidth * box['Left']
                    top = imgHeight * box['Top']
                    width = imgWidth * box['Width']
                    height = imgHeight * box['Height']
                    points = (
                            (left,top),
                            (left + width, top),
                            (left + width, top + height),
                            (left , top + height),
                            (left, top)
                        )
                    draw.line(points, fill=fill_color, width=line_width)

                     # Check if confidence is lower than supplied value
                    if ppe_item['CoversBodyPart']['Confidence'] < confidence:
                        #draw warning yellow bounding box within face mask
bounding box
                        offset=line_width+ line_width
                        points = (
                                    (left+offset,top + offset),
                                    (left + width-offset, top+offset),
                                    ((left) + (width-offset), (top-offset) +
(height)),
                                    (left+ offset , (top) + (height -offset)),
                                    (left + offset, top + offset)
                                )
                        draw.line(points, fill=fill_yellow, width=line_width)

        if found_mask==False:
            # no face mask found so draw red bounding box around body
```

```
            box = person['BoundingBox']
            left = imgWidth * box['Left']
            top = imgHeight * box['Top']
            width = imgWidth * box['Width']
            height = imgHeight * box['Height']
            points = (
                (left,top),
                (left + width, top),
                (left + width, top + height),
                (left , top + height),
                (left, top)
                )
            draw.line(points, fill=fill_red, width=line_width)

    image.show()

def main():
    photo='photo'
    confidence=80
    detect_ppe(photo, confidence)

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

CLI

在下列 CLI 範例中，變更下列引數的值：

- 本機映像檔案 (PNG 或 JPEG) 的 photo 路徑和檔案名稱的值。
- confidence 的值達到所需的可信度等級 (50-100)。
- 將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
                                aws rekognition detect-protective-equipment
--image "{"S3Object":{"Bucket":"bucket-name","Name":"image-name"}}" --profile
profile-name \
                                --summarization-attributes
"{"MinConfidence":MinConfidenceNumber,"RequiredEquipmentTypes":["FACE_COVER"]}"
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
                                aws rekognition detect-protective-equipment  --
image "{\"S3Object\":{\"Bucket\":\"bucket-name\",\"Name\":\"image-name\"}}" \
                                --profile profile-name --summarization-
attributes "{\"MinConfidence\":MinConfidenceNumber,\"RequiredEquipmentTypes\":
[\"FACE_COVER\"]}"
```

辨識名人

Amazon Rekognition 可讓客戶輕鬆使用機器學習自動辨識映像和視頻中數以萬計的知名人物。名人辨識 API 提供的中繼資料可大幅減少標記內容所需的重複性手動工作量，並使其易於搜尋。

由於映像和視頻內容快速擴散，媒體公司經常難以大規模組織、搜索和利用其媒體目錄。為了對當前事件做出回應並創建相關節目，新聞頻道和體育廣播公司通常需要快速找出映像和視頻。中繼資料不足會使這些工作變得困難，但您可以使用 Amazon Rekognition 自動標記大量新內容或封存內容，以便輕鬆搜尋大量廣為人知的國際名人，例如演員、運動員和線上內容創作者。

Amazon Rekognition 名人辨識功能專門適用於已知個別圖像和視頻可能有相關名人的情況。如需辨識非名人臉部的資訊，請參閱 在集合中搜尋人臉。

> **Note**
>
> 如果您是名人且不想加入此功能，請聯絡 Sup AWSport 部門或寄送電子郵件至 <rekognition-celebrity-opt-out@amazon .com>。

主題

- 名人辨識与臉部搜尋比較
- 辨識映像中的名人
- 辨識已儲存影片中的名人
- 取得名人的相關資訊

名人辨識与臉部搜尋比較

Amazon Rekognition 提供名人辨識和臉部辨識功能。這些功能在其使用案例和最佳實務中存在主要差異。

名人辨識已經過預先訓練而能夠辨識各種領域 (例如運動、媒體、政界及商場) 中成千上萬的名人。此功能旨在協助您搜尋大量映像或影片，以識別可能包含特定名人的小集。此功能不是用來比對名人以外群体的人臉。在正確比對名人很重要的情況下，我們建議也使用人工操作員來查看少量標記內容，以協助確保高準確性，並適當應用人類的判斷。使用名人辨識的方式不應該對名人造成負面影響。

另一方面，臉部辨識是更通用的功能，可讓您建立自己的臉部集合與專屬的臉部向量，以驗證身分或搜尋任何人員 (不只是名人)。臉部辨識可以用於驗證大樓進出、公共安全和社交媒體之類的應用。在所有

這些情況下，建議您使用最佳實務、適當的可信度閾值 (包括 99%，適用於公共安全使用案例)，以及在正確比對很重要之情況下的人工檢閱。

如需詳細資訊，請參閱 在集合中搜尋人臉。

辨識映像中的名人

若要辨識影像中的名人並取得已辨識之名人的其他資訊，請使用 RecognizeCelebrities 非儲存 API 操作。例如，在社交媒體或新聞與娛樂業中，時間對資訊收集而言至關重要，您可以使用 `RecognizeCelebrities` 操作來識別映像中多達 100 位名人，並傳回名人的網頁連結 (如果有)。Amazon Rekognition 不會記得在哪張映像中偵測到名人。您的應用程式必須存放此資訊。

如果您尚未存放 `RecognizeCelebrities` 傳回的名人其他資訊，又不想重新分析影像來取得該資訊，請使用 GetCelebrityInfo。若要呼叫 `GetCelebrityInfo`，您需要 Amazon Rekognition 指派給每位名人的唯一識別符。對於映像中已辨識的每位名人，都會在 `RecognizeCelebrities` 回應中傳回此識別符。

如果您要處理大型映像集合以辨識名人，請考慮使用 AWS Batch 在背景批次處理 `RecognizeCelebrities` 的呼叫。當您將新的映像新增至集合時，可以使用 AWS Lambda 函數辨識名人，方法是在映像上傳至 S3 儲存貯體時，呼叫 `RecognizeCelebrities`。

呼叫 RecognizeCelebrities

您可以使用 AWS Command Line Interface (AWS CLI) 或 AWS 開發套件，以映像位元組陣列 (Base64 編碼映像位元組) 或 Amazon S3 物件的方式提供輸入映像。在 AWS CLI 程序中，您會將 .jpg 或 .png 格式的映像上傳至 S3 儲存貯體。在 AWS SDK 程序中，您會使用從本機檔案系統載入的映像。如需輸入映像建議的資訊，請參閱 使用映像。

若要執行此程序，您需要含有一或多個名人臉部的映像檔。

辨識映像中的名人

1. 如果您尚未執行：
   a. 建立或更新具有 `AmazonRekognitionFullAccess` 和 `AmazonS3ReadOnlyAccess` 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。
   b. 安裝並配置 AWS CLI 和 AWS SDKs。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用以下範例來呼叫 `RecognizeCelebrities` 操作。

Java

此範例顯示與映像中偵測到的名人有關的資訊。

將 photo 的值變更為某個映像檔案的路徑和檔案名稱，而該映像檔案含有一個或多個名人臉部。

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

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.Celebrity;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesRequest;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesResult;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import com.amazonaws.util.IOUtils;
import java.util.List;


public class RecognizeCelebrities {

   public static void main(String[] args) {
      String photo = "moviestars.jpg";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ByteBuffer imageBytes=null;
      try (InputStream inputStream = new FileInputStream(new File(photo))) {
         imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
      }
      catch(Exception e)
      {
          System.out.println("Failed to load file " + photo);
```

```
            System.exit(1);
        }


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

        System.out.println("Looking for celebrities in image " + photo + "\n");

        RecognizeCelebritiesResult
 result=rekognitionClient.recognizeCelebrities(request);

        //Display recognized celebrity information
        List<Celebrity> celebs=result.getCelebrityFaces();
        System.out.println(celebs.size() + " celebrity(s) were recognized.\n");

        for (Celebrity celebrity: celebs) {
            System.out.println("Celebrity recognized: " + celebrity.getName());
            System.out.println("Celebrity ID: " + celebrity.getId());
            BoundingBox boundingBox=celebrity.getFace().getBoundingBox();
            System.out.println("position: " +
               boundingBox.getLeft().toString() + " " +
               boundingBox.getTop().toString());
            System.out.println("Further information (if available):");
            for (String url: celebrity.getUrls()){
               System.out.println(url);
            }
            System.out.println();
         }
         System.out.println(result.getUnrecognizedFaces().size() + " face(s) were
 unrecognized.");
      }
}
```

Java V2

此代碼取自AWS文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.recognize_celebs.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
```

```
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesRequest;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Celebrity;
//snippet-end:[rekognition.java2.recognize_celebs.import]

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

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage>\n\n" +
         "Where:\n" +
         "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

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

     String sourceImage = args[0];
     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
```

```
        .build();

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

// snippet-start:[rekognition.java2.recognize_celebs.main]
public static void recognizeAllCelebrities(RekognitionClient rekClient, String
sourceImage) {

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

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

        RecognizeCelebritiesResponse result =
rekClient.recognizeCelebrities(request) ;
        List<Celebrity> celebs=result.celebrityFaces();
        System.out.println(celebs.size() + " celebrity(s) were recognized.\n");
        for (Celebrity celebrity: celebs) {
            System.out.println("Celebrity recognized: " + celebrity.name());
            System.out.println("Celebrity ID: " + celebrity.id());

            System.out.println("Further information (if available):");
            for (String url: celebrity.urls()){
                System.out.println(url);
            }
            System.out.println();
        }
        System.out.println(result.unrecognizedFaces().size() + " face(s) were
unrecognized.");

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

```
 }
 // snippet-end:[rekognition.java2.recognize_celebs.main]
}
```

AWS CLI

此 AWS CLI 命令顯示 `recognize-celebrities` CLI 操作的 JSON 輸出。

將 `bucketname` 變更為 Amazon S3 儲存貯體的名稱，而該儲存貯體含有映像。將 `input.jpg` 變更為某個映像的檔案名稱，而該映像中含有一個或多個名人臉孔。

以您開發人員設定檔的名稱取代 `profile_name` 的值。

```
aws rekognition recognize-celebrities \
  --image "S3Object={Bucket=bucketname,Name=input.jpg}"
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
                                    aws rekognition recognize-celebrities --
image \
                                    "{\"S3Object\":{\"Bucket\":\"bucket-name\",
\"Name\":\"image-name\"}}" --profile profile-name
```

Python

此範例顯示與映像中偵測到的名人有關的資訊。

將 `photo` 的值變更為某個映像檔案的路徑和檔案名稱，而該映像檔案含有一個或多個名人臉部。

將建立 Rekognition 工作階段的行 `profile_name` 中的值取代為您的開發人員設定檔的名稱。

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

import boto3
```

```
def recognize_celebrities(photo):

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

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

    print('Detected faces for ' + photo)
    for celebrity in response['CelebrityFaces']:
        print('Name: ' + celebrity['Name'])
        print('Id: ' + celebrity['Id'])
        print('KnownGender: ' + celebrity['KnownGender']['Type'])
        print('Smile: ' + str(celebrity['Face']['Smile']['Value']))
        print('Position:')
        print('   Left: ' + '{:.2f}'.format(celebrity['Face']['BoundingBox']
['Height']))
        print('   Top: ' + '{:.2f}'.format(celebrity['Face']['BoundingBox']
['Top']))
        print('Info')
        for url in celebrity['Urls']:
            print('   ' + url)
        print()
    return len(response['CelebrityFaces'])

def main():
    photo = 'photo-name'
    celeb_count = recognize_celebrities(photo)
    print("Celebrities detected: " + str(celeb_count))

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

Node.Js

此範例顯示與映像中偵測到的名人有關的資訊。

將 photo 的值變更為某個映像檔案的路徑和檔案名稱，而該映像檔案含有一個或多個名人臉孔。將 bucket 的值變更為包含映像檔案的 S3 儲存貯體。將 REGION 的值變更為與您的使用者相關聯的區域名稱。將建立 Rekognition 工作階段的行 profile_name 中的值取代為您的開發人員設定檔的名稱。

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

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

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

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

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

const recognize_celebrity = async() => {
    try {
        const response = await rekogClient.send(new
 RecognizeCelebritiesCommand(params));
        console.log(response.Labels)
        response.CelebrityFaces.forEach(celebrity =>{
            console.log(`Name: ${celebrity.Name}`)
            console.log(`ID: ${celebrity.Id}`)
            console.log(`KnownGender: ${celebrity.KnownGender.Type}`)
            console.log(`Smile: ${celebrity.Smile}`)
            console.log('Position: ')
            console.log(`   Left: ${celebrity.Face.BoundingBox.Height}`)
            console.log(`  Top : ${celebrity.Face.BoundingBox.Top}`)

        })
        return response.length; // For unit tests.
      } catch (err) {
```

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

recognize_celebrity()
```

.NET

此範例顯示與映像中偵測到的名人有關的資訊。

將 photo 的值變更為某個映像檔案的路徑和檔案名稱，而該映像檔案含有一個或多個名人臉部 (.jpg 或 .png 格式)。

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

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

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

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        RecognizeCelebritiesRequest recognizeCelebritiesRequest = new
 RecognizeCelebritiesRequest();

        Amazon.Rekognition.Model.Image img = new
 Amazon.Rekognition.Model.Image();
        byte[] data = null;
        try
        {
            using (FileStream fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read))
            {
                data = new byte[fs.Length];
```

```
                fs.Read(data, 0, (int)fs.Length);
            }
        }
        catch(Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        img.Bytes = new MemoryStream(data);
        recognizeCelebritiesRequest.Image = img;

        Console.WriteLine("Looking for celebrities in image " + photo + "\n");

        RecognizeCelebritiesResponse recognizeCelebritiesResponse =
rekognitionClient.RecognizeCelebrities(recognizeCelebritiesRequest);

        Console.WriteLine(recognizeCelebritiesResponse.CelebrityFaces.Count + "
celebrity(s) were recognized.\n");
        foreach (Celebrity celebrity in
recognizeCelebritiesResponse.CelebrityFaces)
        {
            Console.WriteLine("Celebrity recognized: " + celebrity.Name);
            Console.WriteLine("Celebrity ID: " + celebrity.Id);
            BoundingBox boundingBox = celebrity.Face.BoundingBox;
            Console.WriteLine("position: " +
               boundingBox.Left + " " + boundingBox.Top);
            Console.WriteLine("Further information (if available):");
            foreach (String url in celebrity.Urls)
                Console.WriteLine(url);
        }
        Console.WriteLine(recognizeCelebritiesResponse.UnrecognizedFaces.Count +
" face(s) were unrecognized.");
    }
}
```

3. 記錄所顯示之其中一個名人 ID 的值。您在 取得名人的相關資訊 中會需要用到。

RecognizeCelebrities 操作請求

`RecognizeCelebrities` 的輸入是映像。在此範例中，映像以映像位元組傳遞。如需詳細資訊，請參閱 使用映像。

```
{
    "Image": {
        "Bytes": "/AoSiyvFpm....."
    }
}
```

RecognizeCelebrities 作業回應

以下是 RecognizeCelebrities 的 JSON 輸入和輸出範例。

RecognizeCelebrities 會傳回已辨識之名人的陣列與無法辨識之臉部的陣列。在範例中，請注意下列事項：

- 已辨識的名人：Celebrities 是已辨識之名人的陣列。陣列中的每個名人物件都會包含名人姓名與指向相關內容的 URL 清單，例如名人的 IMDB 或 Wikidata 連結。Amazon Rekognition 會傳回應用程式可用來判斷名人臉在影像上的位置和名人的唯一識別碼的ComparedFace物件。您之後可透過 GetCelebrityInfo API 操作，使用唯一識別符來擷取名人資訊。
- 無法辨識的臉部：UnrecognizedFaces 是未與任何已知名人相符之臉部的陣列。陣列中的每個 ComparedFace 物件都包含週框方塊 (及其他資訊)，可讓您用來尋找影像中的臉部。

```
{
    "CelebrityFaces": [{
        "Face": {
            "BoundingBox": {
                "Height": 0.617123007774353,
                "Left": 0.15641026198863983,
                "Top": 0.10864841192960739,
                "Width": 0.3641025722026825
            },
            "Confidence": 99.99589538574219,
            "Emotions": [{
                "Confidence": 96.3981749057023,
                "Type": "Happy"
                }
            ],
            "Landmarks": [{
                "Type": "eyeLeft",
                "X": 0.2837241291999817,
                "Y": 0.3637104034423828
```

```
            }, {
                "Type": "eyeRight",
                "X": 0.4091649055480957,
                "Y": 0.37378931045532227
            }, {
                "Type": "nose",
                "X": 0.35267341136932373,
                "Y": 0.49657556414604187
            }, {
                "Type": "mouthLeft",
                "X": 0.2786353826522827,
                "Y": 0.5455248355865479
            }, {
                "Type": "mouthRight",
                "X": 0.39566439390182495,
                "Y": 0.5597742199897766
            }],
            "Pose": {
                "Pitch": -7.749263763427734,
                "Roll": 2.004552125930786,
                "Yaw": 9.012002944946289
            },
            "Quality": {
                "Brightness": 32.69192123413086,
                "Sharpness": 99.9305191040039
            },
            "Smile": {
            "Confidence": 95.45394855702342,
            "Value": True
            }
        },
        "Id": "3Ir0du6",
        "KnownGender": {
            "Type": "Male"
        },
        "MatchConfidence": 98.0,
        "Name": "Jeff Bezos",
        "Urls": ["www.imdb.com/name/nm1757263"]
    }],
    "OrientationCorrection": "NULL",
    "UnrecognizedFaces": [{
        "BoundingBox": {
            "Height": 0.5345501899719238,
            "Left": 0.48461538553237915,
```

```
            "Top": 0.16949152946472168,
            "Width": 0.3153846263885498
        },
        "Confidence": 99.92860412597656,
        "Landmarks": [{
            "Type": "eyeLeft",
            "X": 0.5863404870033264,
            "Y": 0.36940744519233704
        }, {
            "Type": "eyeRight",
            "X": 0.6999204754829407,
            "Y": 0.3769848346710205
        }, {
            "Type": "nose",
            "X": 0.6349524259567261,
            "Y": 0.4804527163505554
        }, {
            "Type": "mouthLeft",
            "X": 0.5872702598571777,
            "Y": 0.5535582304000854
        }, {
            "Type": "mouthRight",
            "X": 0.6952020525932312,
            "Y": 0.5600858926773071
        }],
        "Pose": {
            "Pitch": -7.386096477508545,
            "Roll": 2.304218292236328,
            "Yaw": -6.175624370574951
        },
        "Quality": {
            "Brightness": 37.16635513305664,
            "Sharpness": 99.9305191040039
        },
        "Smile": {
            "Confidence": 95.45394855702342,
            "Value": True
        }
    }]
}
```

辨識已儲存影片中的名人

Amazon Rekognition Video 在已儲存影片中辨識名人是一種非同步操作。若要辨識儲存視訊中的名人，請使StartCelebrityRecognition用開始視訊分析。Amazon Rekognition Video 向其發佈影片分析操作的物件偵測結果和完成狀態的 Amazon Simple Notification Service 主題。如果影片分析成功，請呼叫 GetCelebrityRecognition 以取得分析結果。如需開始視訊分析並取得結果的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

此程序會展開 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中的程式碼，該操作使用 Amazon SQS 佇列來取得影片分析要求的完成狀態。若要執行此程序，您需要含有一或多個名人臉部的影片檔案。

偵測存放在 Amazon S3 儲存貯體 (開發套件) 之影片中的名人

1. 執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。
2. 將下列程式碼新增至您在步驟 1 中建立的類別 `VideoDetect`。

   Java

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

      //
 Celebrities========================================================================
      private static void StartCelebrityDetection(String bucket, String video)
 throws Exception{

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

          StartCelebrityRecognitionRequest req = new
 StartCelebrityRecognitionRequest()
                  .withVideo(new Video()
                        .withS3Object(new S3Object()
                              .withBucket(bucket)
                              .withName(video)))
                  .withNotificationChannel(channel);
```

```
          StartCelebrityRecognitionResult startCelebrityRecognitionResult =
rek.startCelebrityRecognition(req);
          startJobId=startCelebrityRecognitionResult.getJobId();

       }

       private static void GetCelebrityDetectionResults() throws Exception{

          int maxResults=10;
          String paginationToken=null;
          GetCelebrityRecognitionResult celebrityRecognitionResult=null;

          do{
             if (celebrityRecognitionResult !=null){
                paginationToken = celebrityRecognitionResult.getNextToken();
             }
             celebrityRecognitionResult = rek.getCelebrityRecognition(new
GetCelebrityRecognitionRequest()
                   .withJobId(startJobId)
                   .withNextToken(paginationToken)
                   .withSortBy(CelebrityRecognitionSortBy.TIMESTAMP)
                   .withMaxResults(maxResults));


             System.out.println("File info for page");
             VideoMetadata
videoMetaData=celebrityRecognitionResult.getVideoMetadata();

             System.out.println("Format: " + videoMetaData.getFormat());
             System.out.println("Codec: " + videoMetaData.getCodec());
             System.out.println("Duration: " +
videoMetaData.getDurationMillis());
             System.out.println("FrameRate: " + videoMetaData.getFrameRate());

             System.out.println("Job");

             System.out.println("Job status: " +
celebrityRecognitionResult.getJobStatus());


             //Show celebrities
             List<CelebrityRecognition> celebs=
celebrityRecognitionResult.getCelebrities();
```

```
            for (CelebrityRecognition celeb: celebs) {
               long seconds=celeb.getTimestamp()/1000;
               System.out.print("Sec: " + Long.toString(seconds) + " ");
               CelebrityDetail details=celeb.getCelebrity();
               System.out.println("Name: " + details.getName());
               System.out.println("Id: " + details.getId());
               System.out.println();
            }
          } while (celebrityRecognitionResult !=null &&
celebrityRecognitionResult.getNextToken() != null);

       }
```

在函數 `main` 中，將下行:

```
       StartLabelDetection(bucket, video);

       if (GetSQSMessageSuccess()==true)
        GetLabelDetectionResults();
```

取代為：

```
       StartCelebrityDetection(bucket, video);

       if (GetSQSMessageSuccess()==true)
        GetCelebrityDetectionResults();
```

Java V2

此代碼取自AWS文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.recognize_video_celebrity.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
```

```
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.CelebrityRecognitionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.CelebrityRecognition;
import software.amazon.awssdk.services.rekognition.model.CelebrityDetail;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionResponse;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_celebrity.import]

/**
*  To run this code example, ensure that you perform the Prerequisites as stated
 in the Amazon Rekognition Guide:
*  https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
*
* Also, ensure that set up your development environment, including your
 credentials.
*
* For information, see this documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/

public class RecognizeCelebritiesVideo {

private static String startJobId ="";

public static void main(String[] args) {

   final String usage = "\n" +
       "Usage: " +
       "   <bucket> <video> <topicArn> <roleArn>\n\n" +
       "Where:\n" +
       "   bucket - The name of the bucket in which the video is located (for
 example, (for example, myBucket). \n\n"+
```

```
        "   video - The name of video (for example, people.mp4). \n\n" +
        "   topicArn - The ARN of the Amazon Simple Notification Service (Amazon
 SNS) topic. \n\n" +
        "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;

    if (args.length != 4) {
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];
    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

   NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

   StartCelebrityDetection(rekClient, channel, bucket, video);
   GetCelebrityDetectionResults(rekClient);
   System.out.println("This example is done!");
   rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_celebrity.main]
public static void StartCelebrityDetection(RekognitionClient rekClient,
                                            NotificationChannel channel,
                                            String bucket,
                                            String video){
    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();

        Video vidOb = Video.builder()
```

```
            .s3Object(s3Obj)
            .build();

       StartCelebrityRecognitionRequest recognitionRequest =
 StartCelebrityRecognitionRequest.builder()
            .jobTag("Celebrities")
            .notificationChannel(channel)
            .video(vidOb)
            .build();

       StartCelebrityRecognitionResponse startCelebrityRecognitionResult =
 rekClient.startCelebrityRecognition(recognitionRequest);
       startJobId = startCelebrityRecognitionResult.jobId();

   } catch(RekognitionException e) {
       System.out.println(e.getMessage());
       System.exit(1);
   }
}

public static void GetCelebrityDetectionResults(RekognitionClient rekClient) {

   try {
       String paginationToken=null;
       GetCelebrityRecognitionResponse recognitionResponse = null;
       boolean finished = false;
       String status;
       int yy=0 ;

       do{
           if (recognitionResponse !=null)
               paginationToken = recognitionResponse.nextToken();

           GetCelebrityRecognitionRequest recognitionRequest =
 GetCelebrityRecognitionRequest.builder()
               .jobId(startJobId)
               .nextToken(paginationToken)
               .sortBy(CelebrityRecognitionSortBy.TIMESTAMP)
               .maxResults(10)
               .build();

           // Wait until the job succeeds
           while (!finished) {
```

```
                recognitionResponse =
 rekClient.getCelebrityRecognition(recognitionRequest);
                status = recognitionResponse.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
                    Thread.sleep(1000);
                }
                yy++;
            }

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null.
            VideoMetadata videoMetaData=recognitionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<CelebrityRecognition> celebs= recognitionResponse.celebrities();
            for (CelebrityRecognition celeb: celebs) {
                long seconds=celeb.timestamp()/1000;
                System.out.print("Sec: " + seconds + " ");
                CelebrityDetail details=celeb.celebrity();
                System.out.println("Name: " + details.name());
                System.out.println("Id: " + details.id());
                System.out.println();
            }

        } while (recognitionResponse.nextToken() != null);

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.recognize_video_celebrity.main]
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # ============== Celebrities ===============
    def StartCelebrityDetection(self):
        response=self.rek.start_celebrity_recognition(Video={'S3Object':
 {'Bucket': self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetCelebrityDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_celebrity_recognition(JobId=self.startJobId,
                                                MaxResults=maxResults,
                                                NextToken=paginationToken)

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for celebrityRecognition in response['Celebrities']:
                print('Celebrity: ' +
                    str(celebrityRecognition['Celebrity']['Name']))
                print('Timestamp: ' + str(celebrityRecognition['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

在函數 main 中，將下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

取代為：

```
    analyzer.StartCelebrityDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetCelebrityDetectionResults()
```

Node.JS

在下列 Node.Js 程式碼範例中，將 bucket 的值取代為包含您視訊的 S3 儲存貯體的名稱，並以視訊檔案的名稱取代 videoName 的值。您也需要將 roleArn 的值取代為與 IAM 服務角色相關聯的 Arn。最後，將 region 的值替換為與您帳戶關聯的操作區域的名稱。將建立 Rekognition 工作階段的行中 profile_name 的值取代為您開發人員設定檔的名稱。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

// Import required AWS SDK clients and commands for Node.js
import { CreateQueueCommand, GetQueueAttributesCommand, GetQueueUrlCommand,
  SetQueueAttributesCommand, DeleteQueueCommand, ReceiveMessageCommand,
 DeleteMessageCommand } from  "@aws-sdk/client-sqs";
import {CreateTopicCommand, SubscribeCommand, DeleteTopicCommand } from "@aws-
sdk/client-sns";
import  { SQSClient } from "@aws-sdk/client-sqs";
import  { SNSClient } from "@aws-sdk/client-sns";
import  { RekognitionClient, StartLabelDetectionCommand,
 GetLabelDetectionCommand,
  StartCelebrityRecognitionCommand, GetCelebrityRecognitionCommand} from "@aws-
sdk/client-rekognition";
import { stdout } from "process";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
```

```
const profileName = "profile-name"
// Name the collection
// Create SNS service object.
const sqsClient = new SQSClient({ region: REGION,
  credentials: fromIni({profile: profileName,}), });
const snsClient = new SNSClient({ region: REGION,
  credentials: fromIni({profile: profileName,}), });
const rekClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// Set bucket and video variables
const bucket = "bucket-name";
const videoName = "video-name";
const roleArn = "role-arn"
var startJobId = ""

var ts = Date.now();
const snsTopicName = "AmazonRekognitionExample" + ts;
const snsTopicParams = {Name: snsTopicName}
const sqsQueueName = "AmazonRekognitionQueue-" + ts;

 // Set the parameters
 const sqsParams = {
  QueueName: sqsQueueName, //SQS_QUEUE_URL
  Attributes: {
    DelaySeconds: "60", // Number of seconds delay.
    MessageRetentionPeriod: "86400", // Number of seconds delay.
  },
};

const createTopicandQueue = async () => {
  try {
    // Create SNS topic
    const topicResponse = await snsClient.send(new
 CreateTopicCommand(snsTopicParams));
    const topicArn = topicResponse.TopicArn
    console.log("Success", topicResponse);
    // Create SQS Queue
    const sqsResponse = await sqsClient.send(new
 CreateQueueCommand(sqsParams));
    console.log("Success", sqsResponse);
    const sqsQueueCommand = await sqsClient.send(new
 GetQueueUrlCommand({QueueName: sqsQueueName}))
```

```
    const sqsQueueUrl = sqsQueueCommand.QueueUrl
    const attribsResponse = await sqsClient.send(new
 GetQueueAttributesCommand({QueueUrl: sqsQueueUrl, AttributeNames:
 ['QueueArn']}))
    const attribs = attribsResponse.Attributes
    console.log(attribs)
    const queueArn = attribs.QueueArn
    // subscribe SQS queue to SNS topic
    const subscribed = await snsClient.send(new SubscribeCommand({TopicArn:
 topicArn, Protocol:'sqs', Endpoint: queueArn}))
    const policy = {
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "MyPolicy",
          Effect: "Allow",
          Principal: {AWS: "*"},
          Action: "SQS:SendMessage",
          Resource: queueArn,
          Condition: {
            ArnEquals: {
              'aws:SourceArn': topicArn
            }
          }
        }
      ]
    };

    const response = sqsClient.send(new SetQueueAttributesCommand({QueueUrl:
 sqsQueueUrl, Attributes: {Policy: JSON.stringify(policy)}}))
    console.log(response)
    console.log(sqsQueueUrl, topicArn)
    return [sqsQueueUrl, topicArn]

  } catch (err) {
    console.log("Error", err);
  }
};

const startCelebrityDetection = async(roleArn, snsTopicArn) =>{
  try {
      //Initiate label detection and update value of startJobId with returned
 Job ID
```

```
      const response = await rekClient.send(new
 StartCelebrityRecognitionCommand({Video:{S3Object:{Bucket:bucket,
 Name:videoName}},
          NotificationChannel:{RoleArn: roleArn, SNSTopicArn: snsTopicArn}}))
          startJobId = response.JobId
          console.log(`Start Job ID: ${startJobId}`)
          return startJobId
    } catch (err) {
      console.log("Error", err);
    }
  };

const getCelebrityRecognitionResults = async(startJobId) =>{
  try {
      //Initiate label detection and update value of startJobId with returned
 Job ID
      var maxResults = 10
      var paginationToken = ''
      var finished = false

      while (finished == false){
          var response = await rekClient.send(new
 GetCelebrityRecognitionCommand({JobId: startJobId, MaxResults: maxResults,
              NextToken: paginationToken}))
          console.log(response.VideoMetadata.Codec)
          console.log(response.VideoMetadata.DurationMillis)
          console.log(response.VideoMetadata.Format)
          console.log(response.VideoMetadata.FrameRate)
          response.Celebrities.forEach(celebrityRecognition => {
              console.log(`Celebrity: ${celebrityRecognition.Celebrity.Name}`)
              console.log(`Timestamp: ${celebrityRecognition.Timestamp}`)
              console.log()
          })
          // Searh for pagination token, if found, set variable to next token
          if (String(response).includes("NextToken")){
              paginationToken = response.NextToken

          }else{
              finished = true
          }
      }
    } catch (err) {
      console.log("Error", err);
    }
```

```
  };

// Checks for status of job completion
const getSQSMessageSuccess = async(sqsQueueUrl, startJobId) => {
  try {
    // Set job found and success status to false initially
    var jobFound = false
    var succeeded = false
    var dotLine = 0
    // while not found, continue to poll for response
    while (jobFound == false){
      var sqsReceivedResponse = await sqsClient.send(new
 ReceiveMessageCommand({QueueUrl:sqsQueueUrl,
        MaxNumberOfMessages:'ALL', MaxNumberOfMessages:10}));
      if (sqsReceivedResponse){
        var responseString = JSON.stringify(sqsReceivedResponse)
        if (!responseString.includes('Body')){
          if (dotLine < 40) {
            console.log('.')
            dotLine = dotLine + 1
          }else {
            console.log('')
            dotLine = 0
          };
          stdout.write('', () => {
            console.log('');
          });
          await new Promise(resolve => setTimeout(resolve, 5000));
          continue
        }
      }

      // Once job found, log Job ID and return true if status is succeeded
      for (var message of sqsReceivedResponse.Messages){
        console.log("Retrieved messages:")
        var notification = JSON.parse(message.Body)
        var rekMessage = JSON.parse(notification.Message)
        var messageJobId = rekMessage.JobId
        if (String(rekMessage.JobId).includes(String(startJobId))){
          console.log('Matching job found:')
          console.log(rekMessage.JobId)
          jobFound = true
          console.log(rekMessage.Status)
          if (String(rekMessage.Status).includes(String("SUCCEEDED"))){
```

```
            succeeded = true
            console.log("Job processing succeeded.")
            var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
          }
        }else{
          console.log("Provided Job ID did not match returned ID.")
          var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
        }
      }
    }
  return succeeded
  } catch(err) {
    console.log("Error", err);
  }
};

// Start label detection job, sent status notification, check for success
 status
// Retrieve results if status is "SUCEEDED", delete notification queue and
 topic
const runCelebRecognitionAndGetResults = async () => {
  try {
    const sqsAndTopic = await createTopicandQueue();
    //const startLabelDetectionRes = await startLabelDetection(roleArn,
 sqsAndTopic[1]);
    //const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
 startLabelDetectionRes)
    const startCelebrityDetectionRes = await startCelebrityDetection(roleArn,
 sqsAndTopic[1]);
    const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
 startCelebrityDetectionRes)
    console.log(getSQSMessageSuccess)
    if (getSQSMessageSuccess){
      console.log("Retrieving results:")
      const results = await
 getCelebrityRecognitionResults(startCelebrityDetectionRes)
    }
    const deleteQueue = await sqsClient.send(new DeleteQueueCommand({QueueUrl:
 sqsAndTopic[0]}));
```

```
    const deleteTopic = await snsClient.send(new DeleteTopicCommand({TopicArn:
 sqsAndTopic[1]}));
    console.log("Successfully deleted.")
  } catch (err) {
    console.log("Error", err);
  }
};

runCelebRecognitionAndGetResults()
```

CLI

執行以下 AWS CLI 命令來開始偵測影片中的標籤。

```
aws rekognition start-celebrity-recognition --video "{"S3Object":
{"Bucket":"bucket-name","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
--region region-name --profile profile-name
```

更新下列的值：

- 將 bucket-name 與 video-name 變更為您在步驟 2 中指定的 Amazon S3 儲存貯體與視訊檔名稱。
- 將 region-name 變更為您正在使用的 AWS 區域。
- 使用您開發人員設定檔的名稱取代 profile-name 的值。
- 將 topic-ARN 變更為您在 設定 Amazon Rekognition Video 中的步驟 3 建立的 Amazon SNS 主題的 ARN。
- 將 role-ARN 變更為您在步驟 7 建立的 設定 Amazon Rekognition Video 服務角色的 ARN。

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。如需範例，請參閱下列內容：

```
aws rekognition start-celebrity-recognition --video "{\"S3Object\":{\"Bucket\":
\"bucket-name\",\"Name\":\"video-name\"}}" \
--notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
--region region-name --profile profile-name
```

執行正在進行的程式碼範例之後，複製傳回的程式碼 jobID，並將其提供給下列 GetCelebrityRecognition 命令，以 job-id-number 取代您之前收到的 jobID 結果：

```
aws rekognition get-celebrity-recognition --job-id job-id-number --profile
 profile-name
```

> **Note**
>
> 如果您已執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 以外的視訊範例，要取代的程式碼可能會不同。

3. 執行程式碼。在影片中辨識出之名人的資訊便會顯示。

GetCelebrityRecognition 作業回應

以下是 JSON 回應的範例。回應包含下列內容：

- 已辨識的名人：Celebrities 是影片中已辨識之名人的陣列與辨識時間。每當在影片中辨識出該名人時，就會有一個 CelebrityRecognition 物件。每個 CelebrityRecognition 包含經識別名人的資訊 (CelebrityDetail) 和名人在影片中識別的時間 (Timestamp)。會在影片開頭量測 Timestamp (單位：毫秒)。
- CelebrityDetail— 包含有關公認名人的信息。其包括名人姓名 (Name)、識別符 (ID)、名人的已知性別 (KnownGender) 與指向相關內容的 URL 清單 (Urls)。它還包括 Amazon Rekognition Video 在識別準確性方面具有的信心水平，以及有關名人臉部的詳細信息。FaceDetail如果您之後需要取得相關內容，可以搭配 getCelebrityInfo 使用 ID。
- VideoMetadata— 已分析視訊的相關資訊。

```
{
    "Celebrities": [
        {
            "Celebrity": {
                "Confidence": 0.699999988079071,
                "Face": {
                    "BoundingBox": {
                        "Height": 0.20555555820465088,
```

```
                    "Left": 0.029374999925494194,
                    "Top": 0.22333332896232605,
                    "Width": 0.11562500149011612
                },
                "Confidence": 99.89837646484375,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.06857934594154358,
                        "Y": 0.30842265486717224
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.10396526008844376,
                        "Y": 0.300625205039978
                    },
                    {
                        "Type": "nose",
                        "X": 0.0966852456331253,
                        "Y": 0.34081998467445374
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.075217105448246,
                        "Y": 0.3811396062374115
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.10744428634643555,
                        "Y": 0.37407416105270386
                    }
                ],
                "Pose": {
                    "Pitch": -0.9784082174301147,
                    "Roll": -8.808176040649414,
                    "Yaw": 20.28228759765625
                },
                "Quality": {
                    "Brightness": 43.312068939208984,
                    "Sharpness": 99.9305191040039
                }
            },
            "Id": "XXXXXX",
            "KnownGender": {
```

```
                    "Type": "Female"
                },
                "Name": "Celeb A",
                "Urls": []
            },
            "Timestamp": 367
        },......
    ],
    "JobStatus": "SUCCEEDED",
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw==",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

取得名人的相關資訊

在這些程序中，您會使用 getCelebrityInfo API 操作來取得名人資訊。使用之前 RecognizeCelebrities 呼叫所傳回的名人 ID，即可識別名人。

呼叫 GetCelebrityInfo

這些程序也需要 Amazon Rekognition 已知名人的名人 ID。使用您在「辨識映像中的名人」中記下的名人 ID。

取得名人資訊 (開發套件)

1. 如果您尚未執行：

    a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

    b. 安裝及設定 AWS CLI 與 AWS SDK。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 使用以下範例來呼叫 GetCelebrityInfo 操作。

Java

此範例顯示與名人相關的名稱和資訊。

將 id 取代為「辨識映像中的名人」中所顯示的其中一個名人 ID。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.GetCelebrityInfoRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityInfoResult;


public class CelebrityInfo {

   public static void main(String[] args) {
      String id = "nnnnnnnn";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      GetCelebrityInfoRequest request = new GetCelebrityInfoRequest()
         .withId(id);

      System.out.println("Getting information for celebrity: " + id);

      GetCelebrityInfoResult
 result=rekognitionClient.getCelebrityInfo(request);

      //Display celebrity information
      System.out.println("celebrity name: " + result.getName());
      System.out.println("Further information (if available):");
      for (String url: result.getUrls()){
         System.out.println(url);
      }
   }
}
```

Java V2

此代碼取自AWS文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CelebrityInfo {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <id>

                Where:
                   id - The id value of the celebrity. You can use the
 RecognizeCelebrities example to get the ID value.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String id = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
```

```
                .build();

        getCelebrityInfo(rekClient, id);
        rekClient.close();
    }

    public static void getCelebrityInfo(RekognitionClient rekClient, String id)
 {
        try {
            GetCelebrityInfoRequest info = GetCelebrityInfoRequest.builder()
                    .id(id)
                    .build();

            GetCelebrityInfoResponse response =
 rekClient.getCelebrityInfo(info);
            System.out.println("celebrity name: " + response.name());
            System.out.println("Further information (if available):");
            for (String url : response.urls()) {
                System.out.println(url);
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

AWS CLI

此 AWS CLI 命令顯示 `get-celebrity-info` CLI 操作的 JSON 輸出。將 `ID` 取代為「辨識映像中的名人」中所顯示的其中一個名人 ID。以您開發人員設定檔的名稱取代 `profile-name` 的值。

```
aws rekognition get-celebrity-info --id celebrity-id --profile profile-name
```

Python

此範例顯示與名人相關的名稱和資訊。

將 `id` 取代為「辨識映像中的名人」中所顯示的其中一個名人 ID。將建立 Rekognition 工作階段的行中 `profile_name` 的值取代為您的開發人員設定檔的名稱。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def get_celebrity_info(id):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    # Display celebrity info
    print('Getting celebrity info for celebrity: ' + id)

    response = client.get_celebrity_info(Id=id)

    print(response['Name'])
    print('Further information (if available):')
    for url in response['Urls']:
        print(url)

def main():
    id = "celebrity-id"
    celebrity_info = get_celebrity_info(id)

if __name__ == "__main__":
    main()
```

.NET

此範例顯示與名人相關的名稱和資訊。

將 id 取代為「辨識映像中的名人」中所顯示的其中一個名人 ID。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
public class CelebrityInfo
{
    public static void Example()
    {
        String id = "nnnnnnnn";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        GetCelebrityInfoRequest celebrityInfoRequest = new
 GetCelebrityInfoRequest()
        {
            Id = id
        };

        Console.WriteLine("Getting information for celebrity: " + id);

        GetCelebrityInfoResponse celebrityInfoResponse =
 rekognitionClient.GetCelebrityInfo(celebrityInfoRequest);

        //Display celebrity information
        Console.WriteLine("celebrity name: " + celebrityInfoResponse.Name);
        Console.WriteLine("Further information (if available):");
        foreach (String url in celebrityInfoResponse.Urls)
            Console.WriteLine(url);
    }
}
```

GetCelebrityInfo 操作請求

以下是 `GetCelebrityInfo` 的 JSON 輸入和輸出範例。

輸入 `GetCelebrityInfo` 的是所需名人的 ID。

```
{
    "Id": "nnnnnnn"
}
```

GetCelebrityInfo 作業回應

`GetCelebrityInfo` 將連結的陣列 (`Urls`) 傳回至有關所要求之名人的相關資訊。

```
{
    "Name": "Celebrity Name",
    "Urls": [
        "www.imdb.com/name/nmnnnnnnn"
    ]
}
```

管制內容

您可以使用 Amazon Rekognition 偵測不適當、不需要或冒犯性的內容。您可以在社交媒體、廣播媒體、廣告和電子商務環境中使用 Rekognition 管制 API，以建立更安全的使用者體驗、為廣告客戶提供品牌安全保證，並遵守當地和全球法規。

如今，許多公司完全依賴人類審核員來審查第三方或使用者生成的內容，而其他公司則只是對使用者投訴做出反應，以消除令人反感或不適當的映像、廣告或影片。但是，人類審核員無法以充分的質量或速度進行擴展來滿足這些需求，這會導致使用者體驗不佳，實現規模的高成本，甚至損失品牌聲譽。透過使用 Rekognition 審核映像和影片，人類審核員可以檢閱較少內容，通常是機器學習標記的總體容量的 1% 至 5%。這使人類審核員能夠專注於更有價值的活動，並且仍然以現有成本的一小部分完成所有內容管制。若要組建人類審核員並執行人工審核任務，您可以使用已與 Rekognition 整合的 Amazon 增強版 AI。

您可以使用自訂管制特徵來增強協調深度學習模型的準確性。使用自訂管制，您可以上傳映像並註解這些映像來訓練自訂管制轉接器。然後，可以將經過訓練的介面卡提供給「DetectModeration標籤」作業，以增強其在影像上的效能。如需詳細資訊，請參閱透過自訂管制提升準確性。

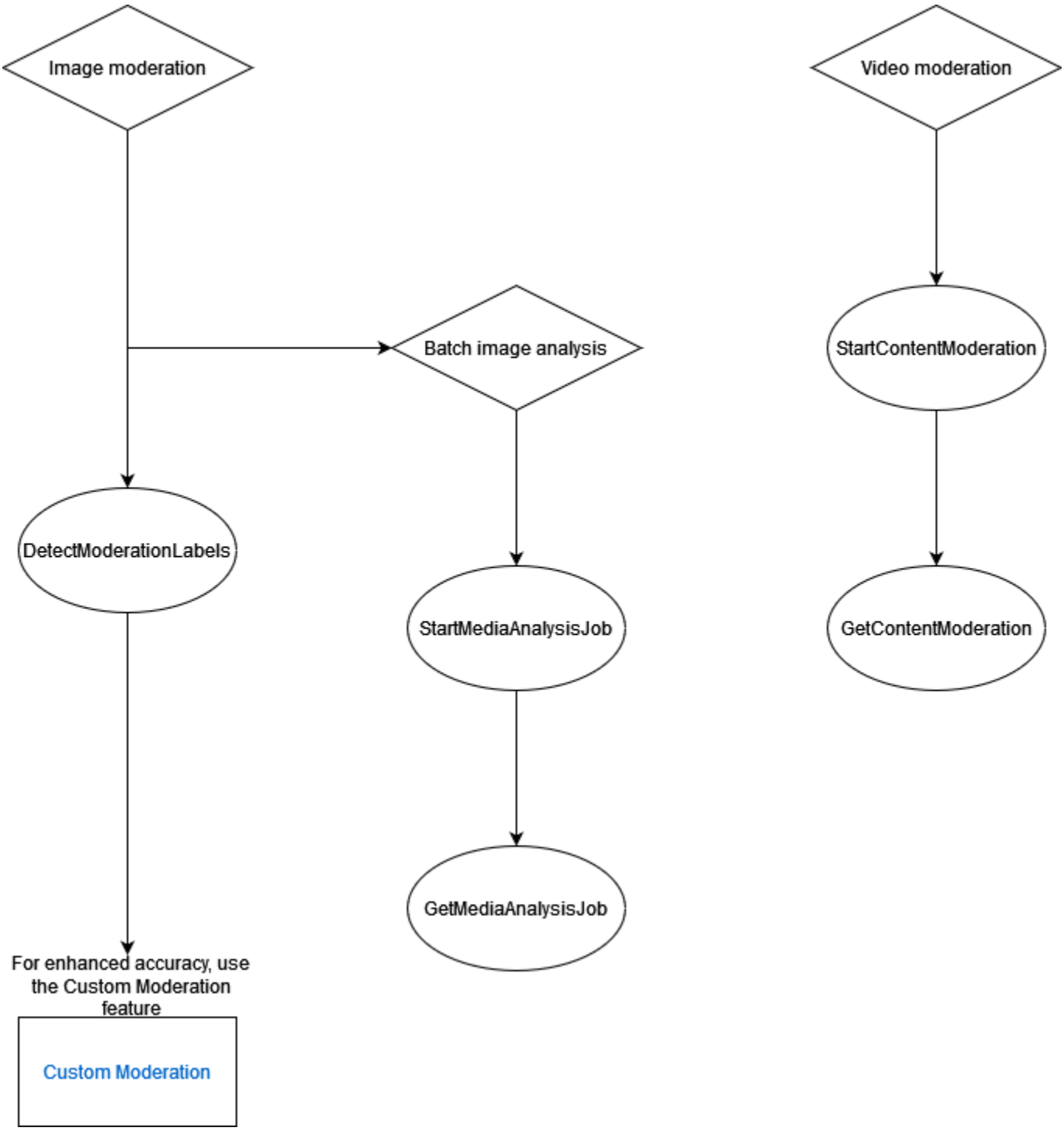
Rekognition 內容協調作業支援的標籤

- 若要下載協調標籤清單，請按一下這裡。

主題

- 使用映像和影片管制 API
- 測試內容協調版本 7 並轉換 API 回應
- 偵測不適當的映像
- 偵測不當儲存的影片
- 透過自訂管制提升準確性
- 使用 Amazon Augmented AI 檢閱不適當的內容

下圖顯示呼叫作業的順序，視您使用內容協調的影像或視訊元件的目標而定：



使用映像和影片管制 API

在 Amazon Rekognition 影像 API 中，您可以使用DetectModeration標籤同步偵測不適當、不需要或冒犯性的內容，並以非同步方式使用和操作。StartMediaAnalysisJobGetMediaAnalysisJob您可以使

用 Amazon Rekognition Video API，透過使用協調和StartContent協調作業以非同步方式偵測此類內容。GetContent

標籤類別

Amazon Rekognition 使用三層階層分類法來標示不適當、不需要或令人反感的內容類別。每個含分類第 1 級 (L1) 的標籤都有許多分類第 2 級標籤 (L2)，而某些分類第 2 級標籤可能有分類第 3 級標籤 (L3)。這允許內容的分層分類。

對於每個偵測到的協調標籤，API 也會傳回`TaxonomyLevel`，其中包含標籤所屬的層級 (1、2 或 3)。例如，圖像可以根據以下分類進行標記：

L1：私密部位和接吻的非露骨裸露，L2：非露骨裸露，L3：暗示裸露。

> **Note**
>
> 我們建議您使用 L1 或 L2 類別來協調您的內容，並且只使用 L3 類別來移除您不想審核的特定概念 (亦即根據您的協調政策，偵測您可能不想分類為不適當、不想要或冒犯性內容的內容)。

下表顯示品類層次與每個層次可能標籤之間的關係。若要下載協調標籤清單，請按一下這裡。

最上層類別 (L1)	二級組別 (L2)	第三級組別 (L3)	定義
明確	露骨裸露	暴露的男性生殖器	男性生殖器，包括陰莖（無論是直立或鬆弛），陰囊和任何可辨別的陰毛。本術語適用於涉及性活動或任何視覺內容的情況下，男性生殖器完全或部分顯示。
		暴露的女性生殖器	女性生殖系統的外部部位，包括外陰，陰道和任何可觀察到的陰毛。本術語適用於涉及性活動或任何視覺內容的情況，其中

			女性解剖學的這些方面完全或部分顯示。
		暴露的臀部或肛門	人的臀部或肛門，包括臀部裸體的情況下，或者通過透明的衣服可以看出它們。該定義特別適用於臀部或肛門直接和完全可見的情況，但不包括任何形式的內衣或衣服提供完整或部分覆蓋的情況。
		裸露的女性乳頭	女性乳頭，包括完全可見和部分可見的 Aerola（乳頭周圍區域）和乳頭。
	明確的性活動	N/A	描述實際或模擬的性行為，包括人類性交、口交，以及男性生殖器刺激和其他身體部位和物體刺激女性生殖器。該術語還包括射精或陰道液體部位和色情做法或角色扮演涉及束縛, 紀律, 統治地位和屈服, 和施虐受虐狂.
	性玩具	N/A	用於性刺激或愉悅的物體或裝置，例如假陽具、振動器、對接插頭、節拍等。

親密部位和接吻的非露骨裸露	非露骨裸露	裸露的背部	從頸部到脊柱末端都可以看到大部分皮膚的人後部。當個人的背部部分或完全閉塞時，此術語不適用。
		外露的男性乳頭	男性乳頭，包括部分可見的乳頭。
		臀部部分暴露	部分暴露的人臀部。本術語包括由於短衣或肛門裂縫部分可見的臀部或臀部臉頰部分可見的部分可見區域。該術語不適用於臀部完全裸露的情況。
		女性乳房部分暴露	部分暴露的人體女性乳房，其中女性乳房的一部分可見或發現，而不會透露整個乳房。本術語適用於當內部乳房褶皺的區域是可見的，或當下乳房摺痕是可見的，奶嘴完全覆蓋或閉塞.
		暗示裸露	一個人誰是裸體, 無論是無上身或無底, 但與親密的部位，如臀部, 乳頭, 或生殖器覆蓋, 閉塞, 或不完全可見.

	受阻的私密部位	女性乳頭受阻	視覺描繪了一種情況，其中女性的乳頭被不透明的衣服或覆蓋物覆蓋，但它們的形狀清晰可見。
		阻塞男性生殖器	視覺描繪的情況下，男性的生殖器或陰莖被不透明的衣服或覆蓋物覆蓋，但其形狀清晰可見。當圖像中受阻的生殖器處於特寫狀態時，此術語適用。
	接吻, 上, the, 嘴唇	N/A	描繪一個人的嘴唇與另一個人的嘴唇接觸。
泳裝或內衣	女性泳裝或內衣	N/A	女性泳裝的人體服裝（例如，一件式泳衣，比基尼，坦基尼等）和女性內衣（例如胸罩，內褲，內褲，內褲，內衣，丁字褲等）
	男性泳裝或內衣	N/A	男性泳裝的人體衣物（例如泳褲、海灘褲、游泳內褲等）和男性內衣（例如內褲、拳擊手等）

暴力	武器	N/A	用於對生物、結構或系統造成傷害或損害的儀器或裝置。這包括槍械（例如槍支、步槍、機槍等）、鋒利的武器（例如劍、刀等）、爆炸物品和彈藥（例如，導彈、炸彈、子彈等）。
	圖形暴力	武器暴力	使用武器對自己、其他個人或財產造成傷害、損害、傷害或死亡。
		身體暴力	對其他個人或財產造成傷害的行為（例如，打擊，打架，拉毛等）或其他涉及人群或多個人的暴力行為。
		自我傷害	對自己造成傷害的行為，通常是通過切割身體部位，例如手臂或腿部，其中切口通常可見。
		血與血腥	暴力對一個人，一組個人或動物的視覺表現，涉及開放性傷口，流血和肢解的身體部位。

		爆炸和爆炸	描繪了強烈的火焰，濃濃的煙霧或灰塵和煙霧從地面噴發出來的暴力和破壞性爆發。
令人反感	死亡與逃亡	異常消瘦的身體	人體是非常薄和營養不良嚴重的身體浪費和肌肉和脂肪組織的消耗.
		屍體	以肢解的身體，懸掛屍體或骷髏形式的人類屍體。
	崩潰	空中墜毀	飛機、直升機或其他飛行車輛等航空車輛的事故，造成損壞、傷害或死亡。本術語適用於空中車輛的部分可見時。
毒品和煙草	產品	藥丸	體積小，堅固，通常是圓形或橢圓形的桌子或膠囊。這個術語適用於以非標準的藥丸，瓶子或透明包裝的形式呈現，並且不適用於服用藥丸的人的視覺描繪。
	毒品、煙草用具及使用	吸煙	吸入，呼氣和照亮燃燒物質，包括香煙，雪茄，電子煙，水煙或聯合的行為。

酒精	酒精使用	飲酒	從瓶子或酒精或烈酒杯中喝酒精飲料的行為。
	含酒精飲品	N/A	關閉一個或多個瓶子的酒精或白酒，玻璃杯或杯子與酒精或酒類，以及由個人持有的酒精或酒類的玻璃杯或杯子。本術語不適用於從瓶裝或杯酒或烈酒中飲用的個人。
粗魯的手勢	中指	N/A	用中指對手勢的視覺描繪向上延伸，而另一根手指則向下折疊。
賭博	N/A	N/A	參與有機會在賭場贏得獎品的遊戲的行為，例如撲克牌，黑傑克，輪盤賭，賭場的老虎機等。
仇恨符號	納粹黨	N/A	與納粹黨相關的符號、旗幟或手勢的視覺描繪。
	白人至上	N/A	與 Ku Klux 三生黨（KKK）相關的符號或衣服以及帶有同盟國旗的圖像的視覺描繪。
	極端主義者	N/A	包含極端主義者和恐怖組織旗幟的圖像。

並非 L2 類別中的每個標籤在 L3 類別中都有支援的標籤。此外，在「產品」和「毒品和煙草用具和用途」L2 標籤下的 L3 標籤並不詳盡無遺。這些 L2 標籤涵蓋上述 L3 標籤以外的概念，在這種情況下，API 回應中只會傳回 L2 標籤。

您可以確定應用程式內容是否適合。例如，可接受映像本身具有暗示性內容，但不接受含有裸露內容。要過濾圖像，請使用由`DetectModerationLabels`（圖像）和`GetContentModeration`（視頻）返回的 label 數組。[ModerationLabel]

內容類型

API 還可以識別動畫或插圖內容類型，並將內容類型作為響應的一部分返回：

- 動畫內容包括電子遊戲和動畫（例如卡通，漫畫，漫畫，動漫）。
- 插圖內容包括繪畫，繪畫和素描。

可信度

您可指定 `MinConfidence` 輸入參數，以設定 Amazon Rekognition 用於偵測不安全內容的可信度閾值。系統不會對偵測到可信度比 `MinConfidence` 低的不安全內容傳回標籤。

指定小於 50% `MinConfidence` 的值可能會傳回大量的假陽性結果 (亦即較高的回復率、較低的精確度)。另一方面，指定 50% `MinConfidence` 以上可能會傳回較少數量的假陽性結果 (即較低的回收率、較高的精確度)。如果您不指定 `MinConfidence` 的值，Amazon Rekognition 會以至少 50% 的可信度，傳回偵測到的不安全內容標籤。

`ModerationLabel` 陣列包含前述類別中的標籤，以及對已辨識內容之正確性所估計的可信度。最上層標籤會與任何已識別的第二層標籤一起傳回。例如，Amazon Rekognition 傳回的最上層標籤可能是具有高信賴分數的「露骨裸露」。這可能足以滿足您的篩選需求。不過，如果有必要，您也可以使用第二層標籤的可信度分數 (例如「男性裸露映像」)，來取得更精細的篩選。如需範例，請參閱[偵測不適當的映像]。

版本控制

Amazon Rekognition Image 和 Amazon Rekognition Video 都會傳回用於偵測不適當內容的協調偵測模型版本 (`ModerationModelVersion`)。

排序和彙總

使用擷取結果時 GetContentModeration，您可以排序和彙總結果。

排序順序：傳回的標籤陣列會依時間排序。若要依標籤排序，請在 SortBy 輸入參數中指定 NAME 以執行 GetContentModeration。如果標籤在視訊中多次出現，將會有 ModerationLabel 元素的多個執行個體。

標籤資訊 — ModerationLabels 陣列元素包含一個ModerationLabel物件，其中包含標籤名稱，以及 Amazon Rekognition 對偵測到標籤準確性的信賴度。時間戳記是偵測到 ModerationLabel 的時間，定義為自影片開始以來經過的毫秒數。對於依影片 SEGMENTS 彙總的結果，會傳回 StartTimestampMillis、EndTimestampMillis 和 DurationMillis 的結構，分別定義區段的開始時間、結束時間和持續時間。

彙總：指定傳回時如何彙總結果。依據 TIMESTAMPS 彙總預設值。您也可以選擇依據 SEGMENTS 彙總，以便在時間範圍內彙總結果。僅傳回區段期間偵測到的標籤。

自訂協調配接卡狀態

自訂協調配接器可以處於下列其中一種狀態：訓練 _ 進度、訓練 _ 完成、訓練 _ 失敗、刪除、已取代或已過期。如需這些轉接器狀態的完整說明，請參閱管理轉接器。

> **Note**
>
> Amazon Rekognition 不是授權機構，也不會以任何方式聲稱是不適當或令人反感的內容的詳盡篩選條件。此外，映像和影片審核 API 不會偵測映像是否包含非法內容，例如兒童色情。

測試內容協調版本 7 並轉換 API 回應

Rekognition 將內容協調標籤偵測功能之影像視訊元件的機器學習模型從 6.1 版更新至 7 版。此更新提高了整體準確性，並引入了幾個新類別以及修改其他類別。

如果您目前是 6.1 版的影片使用者，建議您採取下列動作，以順暢轉換至第 7 版：

1. 下載並使用 AWS 私有開發套件 (請參閱the section called “AWS 內容協調第 7 版的 SDK 和使用指南 ”) 來呼叫 StartContentModeration API。
2. 檢閱 API 回應或主控台中傳回的標籤和可信度分數的更新清單。視需要調整應用程式後處理邏輯。
3. 您的帳戶將保留為 6.1 版，直到 2024 年 5 月 13 日為止。如果您希望在 2024 年 5 月 13 日之後使用 6.1 版，請在 2024 年 4 月 30 日前聯絡 AWS Support 團隊以申請延期。我們可以將您的帳戶延長至 6.1 版，直到 2024 年 6 月 10 日為止。如果我們在 2024 年 4 月 30 日之前仍未收到您的回覆，您的帳戶將自動從 2024 年 5 月 13 日起遷移至 7.0 版。

AWS 內容協調第 7 版的 SDK 和使用指南

下載與您選擇的開發語言對應的 SDK，並參閱適當的使用者指南。

開發套件的連結	安裝/使用指南
爪哇 1.X	指南-爪哇 1.pdf
爪哇 2.x	指南-爪哇 2.pdf
JavaScript V2	指南- JavaScript v2.pdf
JavaScript v3	指南- JavaScript v3.pdf
Python	指南-Python 和 AWS CLI.pdf
Ruby	指南-RubyV3.pdf
去	指南-去 V1.pdf
go_v2	指南-去 V2.pdf
DotNet	指南-.
PHP	指南-PHP.pdf

6.1 至 7 版的標籤對映

內容審核版本 7 新增了新的標示品類，並修改了先前既有的標示名稱。請參考決定如何將 6.1 個標籤對應至 7 個標籤the section called “ 標籤類別 ”時找到的分類表格。

您可以在下一節中找到一些標籤對映範例。建議您先檢閱這些對映和標籤定義，然後再根據應用程式的後處理邏輯進行必要的更新。

L1 對應綱要

如果您使用僅篩選頂層類別 (L1) 的後處理邏輯 (例如 `Explicit NuditySuggestive`，`Violence`等等)，請參閱下表以更新程式碼。

V6.1 L1	V7 L1

露骨裸露	明確
暗示性	親密部位和接吻的非露骨裸露
	泳裝或內衣
暴力	暴力
令人反感	令人反感
粗魯的手勢	粗魯的手勢
毒品	毒品和煙草
煙草	毒品和煙草
酒精	酒精
賭博	賭博
仇恨符號	仇恨符號

L2 對應綱要

如果您使用過濾 L1 和 L2 類別 (例如`Violence / Weapon Violence`等) 的後處理邏輯`Explicit Nudity / Nudity, Suggestive / Female Swimwear Or Underwear`，請參閱下表來更新您的程式碼。

V6.1 L1	V6.1 L2	V7 L1	V7 L2	V7 L3	V7 ContentTypes
露骨裸露	裸露	明確	露骨裸露	裸露的女性乳頭	
				暴露的臀部或肛門	
	男性裸露圖片	明確	露骨裸露	暴露的男性生殖器	

	女性裸露圖片	明確	露骨裸露	暴露的女性生殖器	
	性行為	明確	明確的性活動		
	插圖明的露骨裸體	明確	露骨裸露		地圖到「動畫」和「插圖」
	插圖明的露骨裸體	明確	明確的性活動		地圖到「動畫」和「插圖」
	成人玩具	明確	性玩具		
暗示性	女性泳衣或內衣褲	泳裝或內衣	女性泳裝或內衣		
	男性泳衣或內衣褲	泳裝或內衣	男性泳裝或內衣		
	部分裸露	親密部位和接吻的非露骨裸露	非露骨裸露	暗示裸露	
	赤身男性	親密部位和接吻的非露骨裸露	非露骨裸露	外露的男性乳頭	
	暴露的服裝	親密部位和接吻的非露骨裸露	非露骨裸露		
		親密部位和接吻的非露骨裸露	受阻的私密部位		

	性情況	親密部位和接吻的非露骨裸露	接吻, 上, the, 嘴唇	
暴力	暴力畫面或血腥	暴力	圖形暴力	血與血腥
	身體暴力	暴力	圖形暴力	身體暴力
	武器暴力	暴力	圖形暴力	武器暴力
	武器	暴力	武器	
	自我傷害	暴力	圖形暴力	自我傷害
令人反感	異常消瘦的身體	令人反感	死亡與逃亡	異常消瘦的身體
	屍體	令人反感	死亡與逃亡	屍體
	絞刑	令人反感	死亡與逃亡	屍體
	空中墜毀	令人反感	崩潰	空中墜毀
	爆炸和爆破	暴力	圖形暴力	爆炸和爆炸
粗魯的手勢	中指	粗魯的手勢	中指	
毒品	藥品	毒品和煙草	產品	
	吸毒	毒品和煙草	毒品、煙草用具及使用	
	藥丸	毒品和煙草	產品	藥丸
	毒品用具	毒品和煙草	毒品、煙草用具及使用	
煙草	煙草產品	毒品和煙草	產品	

	吸煙	毒品和煙草	毒品、煙草用具及使用	吸煙
酒精	飲酒	酒精	酒精使用	飲酒
	含酒精飲品	酒精	含酒精飲品	
賭博	賭博	賭博		
仇恨符號	納粹黨	仇恨符號	納粹黨	
	白人至上	仇恨符號	白人至上	
	極端主義者	仇恨符號	極端主義者	

偵測不適當的映像

您可以使用「標DetectModeration籤」操作來判斷影像是否包含不適當或令人反感的內容。如需 Amazon Rekognition 中的管制標籤清單，請參閱使用映像和影片管制 API。

偵測映像中的不適當內容

映像的格式必須是 .jpg 或 .png。您可以用映像位元組陣列 (Base64 編碼映像位元組) 的方式提供輸入映像，或指定 Amazon S3 物件。在這些程序中，您會將映像 (.jpg 或 .png) 上傳至 S3 儲存貯體。

若要執行這些程序，您必須安裝 AWS CLI 或適當的 AWS SDK。如需詳細資訊，請參閱 Amazon Rekognition 入門。您使用的 AWS 帳戶必須有 Amazon Rekognition API 的存取權限。如需詳細資訊，請參閱 Amazon Rekognition 定義的動作。

偵測映像中的管制標籤 (SDK)

1. 如果您尚未執行：

   a. 建立或更新具有 `AmazonRekognitionFullAccess` 和 `AmazonS3ReadOnlyAccess` 權限的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 將映像上傳至您的 S3 儲存貯體。

如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例來呼叫 DetectModerationLabels 操作。

Java

此範例輸出偵測到的不適當內容標籤名稱、信心度以及偵測到的管制標籤的父標籤。

將 bucket 與 photo 的值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像檔案名稱。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.DetectModerationLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectModerationLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.ModerationLabel;
import com.amazonaws.services.rekognition.model.S3Object;

import java.util.List;

public class DetectModerationLabels
{
   public static void main(String[] args) throws Exception
   {
      String photo = "input.jpg";
      String bucket = "bucket";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      DetectModerationLabelsRequest request = new
 DetectModerationLabelsRequest()
        .withImage(new Image().withS3Object(new
 S3Object().withName(photo).withBucket(bucket)))
        .withMinConfidence(60F);
      try
```

```
      {
           DetectModerationLabelsResult result =
 rekognitionClient.detectModerationLabels(request);
           List<ModerationLabel> labels = result.getModerationLabels();
           System.out.println("Detected labels for " + photo);
           for (ModerationLabel label : labels)
           {
              System.out.println("Label: " + label.getName()
               + "\n Confidence: " + label.getConfidence().toString() + "%"
               + "\n Parent:" + label.getParentName());
          }
       }
       catch (AmazonRekognitionException e)
       {
         e.printStackTrace();
       }
    }
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
//snippet-start:[rekognition.java2.detect_mod_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsRequest;
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.ModerationLabel;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_mod_labels.import]
```

```
/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class ModerateLabels {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage>\n\n" +
         "Where:\n" +
         "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

     if (args.length < 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String sourceImage = args[0];
     Region region = Region.US_WEST_2;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     detectModLabels(rekClient, sourceImage);
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.detect_mod_labels.main]
 public static void detectModLabels(RekognitionClient rekClient, String
 sourceImage) {

     try {
         InputStream sourceStream = new FileInputStream(sourceImage);
         SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
```

```
        Image souImage = Image.builder()
            .bytes(sourceBytes)
            .build();

        DetectModerationLabelsRequest moderationLabelsRequest =
DetectModerationLabelsRequest.builder()
            .image(souImage)
            .minConfidence(60F)
            .build();

        DetectModerationLabelsResponse moderationLabelsResponse =
rekClient.detectModerationLabels(moderationLabelsRequest);
        List<ModerationLabel> labels =
moderationLabelsResponse.moderationLabels();
        System.out.println("Detected labels for image");

        for (ModerationLabel label : labels) {
            System.out.println("Label: " + label.name()
                + "\n Confidence: " + label.confidence().toString() + "%"
                + "\n Parent:" + label.parentName());
        }

    } catch (RekognitionException | FileNotFoundException e) {
        e.printStackTrace();
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.detect_mod_labels.main]
```

AWS CLI

此 AWS CLI 命令會顯示 `detect-moderation-labels` CLI 作業的 JSON 輸出。

將 bucket 與 `input.jpg` 取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像檔案名稱。使用您開發人員設定檔的名稱取代 `profile_name` 的值。若要使用轉接器，請將專案版本的 ARN 提供給 `project-version` 參數。

```
aws rekognition detect-moderation-labels --image "{S3Object:{Bucket:<bucket-
name>,Name:<image-name>}}" \
--profile profile-name \
--project-version "ARN"
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
aws rekognition detect-moderation-labels --image "{\"S3Object\":{\"Bucket\":
\"bucket-name\",\"Name\":\"image-name\"}}" \
--profile profile-name
```

Python

此範例輸出偵測到的不適當或令人反感的內容標籤名稱、信心度以及偵測到的不安全內容標籤的父標籤。

在函數 `main` 中，將 `bucket` 與 `photo` 的值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像檔案名稱。將建立 Rekognition 工作階段的行中 `profile_name` 值取代為您開發人員設定檔的名稱。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def moderate_image(photo, bucket):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    response = client.detect_moderation_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}})

    print('Detected labels for ' + photo)
    for label in response['ModerationLabels']:
        print (label['Name'] + ' : ' + str(label['Confidence']))
        print (label['ParentName'])
    return len(response['ModerationLabels'])

def main():

    photo='image-name'
    bucket='bucket-name'
    label_count=moderate_image(photo, bucket)
```

```
    print("Labels detected: " + str(label_count))

if __name__ == "__main__":
    main()
```

.NET

此範例輸出偵測到的令人反感的內容標籤名稱、信心度以及偵測到的管制標籤的父標籤。

將 bucket 與 photo 的值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像檔案名稱。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectModerationLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectModerationLabelsRequest detectModerationLabelsRequest = new
 DetectModerationLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MinConfidence = 60F
        };
```

```
        try
        {
            DetectModerationLabelsResponse detectModerationLabelsResponse =
 rekognitionClient.DetectModerationLabels(detectModerationLabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (ModerationLabel label in
 detectModerationLabelsResponse.ModerationLabels)
                Console.WriteLine("Label: {0}\n Confidence: {1}\n Parent: {2}",
                    label.Name, label.Confidence, label.ParentName);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

DetectModerationLabels 操作請求

DetectModerationLabels 的輸入是映像。在此範例 JSON 輸入中，來源映像從 Amazon S3 儲存貯體載入。MinConfidence 是在偵測到的標籤的準確性中 Amazon Rekognition Image 必須具有的最低可信度，以便在回應中傳回。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MinConfidence": 60
}
```

DetectModerationLabels 作業回應

DetectModerationLabels 可從 S3 儲存貯體擷取輸入映像，您也可以用映像位元組的方式提供輸入映像。下列範例是來自 DetectModerationLabels 呼叫的回應。

在下列範例 JSON 回應中，請注意以下事項：

- 不適當的映像偵測資訊：此範例會顯示映像中發現不當或冒犯性內容的標籤清單。此清單包含最上層標籤與映像中偵測到的每個第二層標籤。

  標籤：每個標籤都具有名稱、Amazon Rekognition 對標籤正確性估計的可信度，以及其父標籤的名稱。最上層標籤的父名稱為 ""。

  標籤可信度：每個標籤都有一個介於 0 到 100 之間的可信度值，該值表示 Amazon Rekognition 对標籤正確性估计的可信度百分比。您可以為要在 API 操作要求回應中傳回的標籤指定要求的可信度。


```
{
    "ModerationLabels": [
        {
            "Confidence": 99.44782257080078,
            "Name": "Smoking",
            "ParentName": "Drugs & Tobacco Paraphernalia & Use",
            "TaxonomyLevel": 3
        },
        {
            "Confidence": 99.44782257080078,
            "Name": "Drugs & Tobacco Paraphernalia & Use",
            "ParentName": "Drugs & Tobacco",
            "TaxonomyLevel": 2
        },
        {
            "Confidence": 99.44782257080078,
            "Name": "Drugs & Tobacco",
            "ParentName": "",
            "TaxonomyLevel": 1
        }
    ],
    "ModerationModelVersion": "7.0",
    "ContentTypes": [
        {
            "Confidence": 99.9999008178711,
            "Name": "Illustrated"
        }
    ]
}
```

偵測不當儲存的影片

Amazon Rekognition Video 在儲存的影片中偵測不當或令人反感的內容是非同步操作。若要開始偵測不適當或令人反感的內容，請致電StartContent協調。Amazon Rekognition Video 向 Amazon Simple Notification Service 主題發佈影片的完成狀態。如果視訊分析成功，請呼叫「GetContent協調」以取得分析結果。如需開始影片分析並取得結果的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。如需 Amazon Rekognition 中的管制標籤清單，請參閱使用映像和影片管制 API。

此程序會展開 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中的程式碼，該操作使用 Amazon Simple Queue Service 佇列來取得影片分析要求的完成狀態。

偵測 Amazon S3 儲存貯體 (SDK) 中存放的影片中的不當或令人反感的內容

1. 執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。
2. 將下列程式碼新增至您在步驟 1 中建立的類別 `VideoDetect`。

   Java

```
    //Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
    //PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

        //Content moderation
 ========================================================================
        private static void StartUnsafeContentDetection(String bucket, String
 video) throws Exception{

            NotificationChannel channel= new NotificationChannel()
                    .withSNSTopicArn(snsTopicArn)
                    .withRoleArn(roleArn);

            StartContentModerationRequest req = new
 StartContentModerationRequest()
                    .withVideo(new Video()
                            .withS3Object(new S3Object()
                                .withBucket(bucket)
                                .withName(video)))
                    .withNotificationChannel(channel);
```

```
            StartContentModerationResult startModerationLabelDetectionResult =
rek.startContentModeration(req);
            startJobId=startModerationLabelDetectionResult.getJobId();

        }

        private static void GetUnsafeContentDetectionResults() throws
Exception{

            int maxResults=10;
            String paginationToken=null;
            GetContentModerationResult moderationLabelDetectionResult =null;

            do{
                if (moderationLabelDetectionResult !=null){
                    paginationToken =
moderationLabelDetectionResult.getNextToken();
                }

                moderationLabelDetectionResult = rek.getContentModeration(
                        new GetContentModerationRequest()
                            .withJobId(startJobId)
                            .withNextToken(paginationToken)
                            .withSortBy(ContentModerationSortBy.TIMESTAMP)
                            .withMaxResults(maxResults));



                VideoMetadata
videoMetaData=moderationLabelDetectionResult.getVideoMetadata();

                System.out.println("Format: " + videoMetaData.getFormat());
                System.out.println("Codec: " + videoMetaData.getCodec());
                System.out.println("Duration: " +
videoMetaData.getDurationMillis());
                System.out.println("FrameRate: " +
videoMetaData.getFrameRate());


                //Show moderated content labels, confidence and detection
times
                List<ContentModerationDetection> moderationLabelsInFrames=
                        moderationLabelDetectionResult.getModerationLabels();
```

```
                for (ContentModerationDetection label:
 moderationLabelsInFrames) {
                    long seconds=label.getTimestamp()/1000;
                    System.out.print("Sec: " + Long.toString(seconds));
                    System.out.println(label.getModerationLabel().toString());
                    System.out.println();
                }
            } while (moderationLabelDetectionResult !=null &&
 moderationLabelDetectionResult.getNextToken() != null);

        }
```

在函數 `main` 中，將下行:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

取代為：

```
        StartUnsafeContentDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetUnsafeContentDetectionResults();
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
```

```
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import
 software.amazon.awssdk.services.rekognition.model.ContentModerationDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectInappropriate {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
```

```
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startModerationDetection(rekClient, channel, bucket, video);
        getModResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startModerationDetection(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {

        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartContentModerationRequest modDetectionRequest =
StartContentModerationRequest.builder()
                    .jobTag("Moderation")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartContentModerationResponse startModDetectionResult = rekClient
                    .startContentModeration(modDetectionRequest);
            startJobId = startModDetectionResult.jobId();
```

```
        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getModResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetContentModerationResponse modDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (modDetectionResponse != null)
                    paginationToken = modDetectionResponse.nextToken();

                GetContentModerationRequest modRequest =
GetContentModerationRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    modDetectionResponse =
rekClient.getContentModeration(modRequest);
                    status = modDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
```

```
                VideoMetadata videoMetaData =
 modDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<ContentModerationDetection> mods =
 modDetectionResponse.moderationLabels();
                for (ContentModerationDetection mod : mods) {
                    long seconds = mod.timestamp() / 1000;
                    System.out.print("Mod label: " + seconds + " ");
                    System.out.println(mod.moderationLabel().toString());
                    System.out.println();
                }

            } while (modDetectionResponse != null &&
 modDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # ============== Unsafe content ===============
    def StartUnsafeContent(self):
        response=self.rek.start_content_moderation(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)
```

```
    def GetUnsafeContentResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_content_moderation(JobId=self.startJobId,
                                                       MaxResults=maxResults,
                                                       NextToken=paginationToken,
                                                       SortBy="NAME",
                                                       AggregateBy="TIMESTAMPS")

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']
['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for contentModerationDetection in response['ModerationLabels']:
                print('Label: ' +
                    str(contentModerationDetection['ModerationLabel']['Name']))
                print('Confidence: ' +
                    str(contentModerationDetection['ModerationLabel']
['Confidence']))
                print('Parent category: ' +
                    str(contentModerationDetection['ModerationLabel']
['ParentName']))
                print('Timestamp: ' +
 str(contentModerationDetection['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

在函數 main 中，將下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

取代為：

```
    analyzer.StartUnsafeContent()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetUnsafeContentResults()
```

> **Note**
>
> 如果您已執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 以外的影片範例，要取代的程式碼可能會不同。

3. 執行程式碼。在影片中偵測到的不安全內容標籤清單便會顯示。

GetContentModeration 作業回應

來自的響應GetContentModeration是一個數組ModerationLabels，ContentModeration檢測對象。此陣列在每次偵測到不安全內容標籤時就會包含一個元素。在ContentModerationDetectionObject物件中，ModerationLabel包含偵測到不適當或冒犯性內容項目的資訊。 Timestamp是從視訊開始處開始的時間 (以毫秒為單位) 偵測到標籤。標籤會按照階層排列，其排列方式和不安全內容映像分析偵測到的標籤相同。如需詳細資訊，請參閱 管制內容。

以下是來自的範例回應 GetContentModeration，依據 NAME 排序，並依據 TIMESTAMPS 彙總。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
        {
            "Timestamp": 36000,
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
```

```
                "Name": "Alcohol",
                "ParentName": "",
                "TaxonomyLevel": 1
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        },
        {
            "Timestamp": 36000,
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
                "Name": "Alcoholic Beverages",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "bucket-name",
            "Name": "video-name.mp4"
        }
    },
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "TIMESTAMPS"
    }
}
```

以下是來自的範例回應 GetContentModeration，依據 NAME 排序，並依據 SEGMENTS 彙總。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
        {
            "Timestamp": 0,
            "ModerationLabel": {
                "Confidence": 0.0003000000142492354,
                "Name": "Alcohol Use",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 29520,
            "DurationMillis": 29520,
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Illustrated"
                },
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "bucket-name",
            "Name": "video-name.mp4"
        }
    },
```

```
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "SEGMENTS"
    }
}
```

透過自訂管制提升準確性

Amazon Rekognition 的DetectModeration標籤 API 可讓您偵測不適當、不需要或冒犯性的內容。Rekognition 自訂協調功能可讓您使用配接器來增強DetectModeration標籤的準確性。轉接器是模組化的元件，可以新增至現有的 Rekognition 深度學習模型，擴充其功能以適用於其所訓練的任務。藉由建立轉接器並將其提供給「標DetectModeration籤」作業，您可以針對與特定使用案例相關的內容協調工作達到更好的準確性。

為特定審核標籤自訂 Rekognition 的內容管制模型時，您必須建立專案並在提供的一組映像上訓練轉接器。然後，您可以反覆檢查轉接器的效能，並將轉接器重新訓練到想要的準確度。專案可包含不同版本的轉接器。

您可使用 Rekognition 主控台來建立專案和轉接器。或者，您可以使用 AWS SDK 和關聯的 API 來建立專案、訓練轉接器，以及管理您的轉接器。

建立和使用轉接器

轉接器是模組化元件，可新增至現有的 Rekognition 深度學習模型中，擴充其功能以適用於其所訓練的任務。透過使用轉接器訓練深度學習模型，您可以使與特定使用案例相關的映像分析工作達到更高的準確度。

若要建立和使用轉接器，您必須提供訓練和測試資料給 Rekognition。您可以透過兩種方式完成此操作：

- 批量分析和驗證：您可以透過批量分析 Rekognition 將分析並指派標籤的映像來建立訓練資料集。然後，您可以查看為映像生成的註釋，並驗證或更正預測。如需映像批量分析如何運作的詳細資訊，請參閱批量分析。
- 手動註釋：使用這種方法，可以透過上傳和註釋映像來建立訓練資料。您可以透過上傳和註釋映像或通過自動分割來建立測試資料。

選擇下列其中一個主題以進一步了解：

主題

批量分析和驗證

透過這種方法，您可以上傳大量要用作訓練資料的映像，然後使用 Rekognition 取得這些映像的預測結果，並自動為這些映像指派標籤。您可以使用這些預測做為轉接器的起點。您可以驗證預測的準確性，然後根據驗證的預測來訓練轉接器。這可以通過 AWS 控制台完成。

批量分析和自訂協調

上傳映像以進行批量分析

若要為您的轉接器建立訓練資料集，請批量上傳映像以供 Rekognition 預測標籤。為了獲得最佳結果，應盡可能提供最多可訓練的映像，最高限制為 10000，並確保映像代表使用案例的各個層面。

使用 AWS 主控台時，您可以直接從電腦上傳影像，或提供可存放影像的 Amazon 簡易儲存服務儲存貯體。但是，將 Rekognition API 與 SDK 搭配使用時，您必須提供資訊清單檔案，該檔案參考存放在 Amazon Simple Storage Service 儲存貯體中的映像。如需詳細資訊，請參閱批量分析。

審查預測

一旦您將映像上傳到 Rekognition 主控台，Rekognition 就會為這些映像產生標籤。然後，您可以將預測驗證為以下類別之一：真陽性、偽陽性、真陰性、偽陰性。驗證預測後，可以根據您的反饋訓練轉接器。

訓練轉接器

完成批量分析傳回的預測驗證後，您可以啟動轉接器的訓練程序。

取得 AdapterId

轉接器訓練完畢後，您就可以取得轉接器的唯一 ID，以便與 Rekognition 的映像分析 API 搭配使用。

呼叫 API 操作

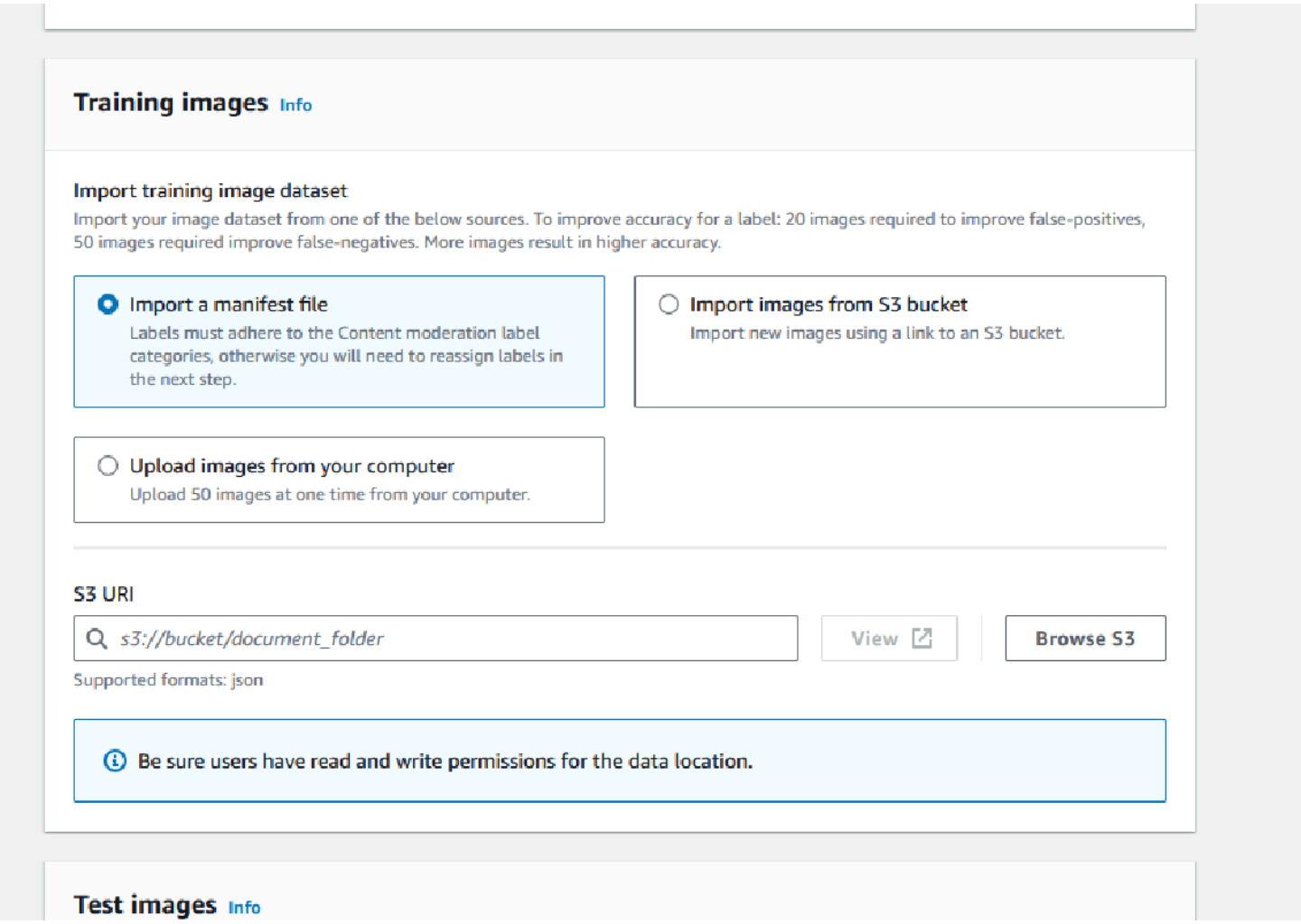
若要套用您的自訂轉接器，請在呼叫其中一個支援轉接器的映像分析 API 時提供其 ID。這樣可以增強映像預測的準確性。

手動註釋

使用這種方法，您可以透過手動上傳和註釋映像來建立訓練資料。您可以透過上傳和註解測試映像或自動分割來建立測試資料，讓 Rekognition 自動使用部分訓練資料做為測試映像。

上傳和註解映像

若要訓練轉接器，您必須上傳一組代表您使用案例的範例映像。為了獲得最佳結果，應盡可能提供最多可訓練的映像，最高限制為 10000，並確保映像代表使用案例的各個層面。



使用 AWS 主控台時，您可以直接從電腦上傳映像、提供資訊清單檔案，或提供可存放映像的 Amazon S3 儲存貯體。

但是，將 Rekognition API 與 SDK 搭配使用時，必須提供清單檔案，該檔案參考儲存在 Amazon S3 儲存貯體中的映像。

您可以使用 Rekognition 主控台的註解介面來為映像加上註解。通過使用標籤標記映像來註釋映像，為培訓確立「基本真相」。您也必須指定訓練和測試集，或使用自動分割特徵，才能訓練轉接器。完成指定資料集並註解映像後，您可以根據測試集中的註解映像建立配接器。然後，您可以評估轉接器的效能。

建立測試集。

您將需要提供帶註釋的測試集或使用自動拆分特徵。訓練組是用來實際訓練轉接器。轉接器學習這些帶註釋的映像中包含的模式。測試集用於在完成轉接器之前評估模型的效能。

訓練轉接器

完成訓練資料的註解或提供資訊清單檔案後，您可以啟動轉接器的訓練串流程。

取得轉接器 ID

轉接器訓練完畢後，您就可以取得轉接器的唯一 ID，以便與 Rekognition 的映像分析 API 搭配使用。

呼叫 API 操作

若要套用您的自訂轉接器，請在呼叫其中一個支援轉接器的映像分析 API 時提供其 ID。這樣可以增強映像預測的準確性。

準備資料集

建立配接器時，您必須提供 Rekognition 兩個資料集，一個訓練資料集和一個測試資料集。每個資料集由提供兩個元素組成：映像和註釋/標籤。以下各節將說明標籤和映像用於哪些標籤和映像，以及其如何結合在一起建立資料集。

映像

您需要在映像的代表性樣本上訓練轉接器。當您選取要訓練的映像時，請嘗試包含至少一些映像，這些映像會顯示您使用轉接器鎖定的每個標籤的預期回應。

若要建立訓練資料集，您必須提供下列兩種映像類型之一：

- 帶有偽陽性預測的映像。例如，當基礎模型預測映像存在酒精時，但實際不存在酒精。
- 帶有偽陰性預測的映像。例如，當基礎模型預測映像不存在酒精時，但實際存在酒精。

若要建立平衡資料集，建議您提供下列兩種映像類型之一：

- 具有真陽性預測的映像。例如，當基礎模型正確預測映像存在酒精。如果您提供偽陰性映像，建議提供這些映像。
- 具有真陽性預測的映像。例如，當基礎模型正確地預測映像不存在酒精時。如果您提供偽陰性映像，建議提供這些映像。

標籤

標籤是指下列任何一項：物件、事件、概念或活動。對於內容審核，標籤是指不適當、不需要或令人反感的內容執行個體。

在將標籤指派給稱為「註釋」的映像時，在透過訓練 Rekognition 的基礎模型來建立轉接器的環境。使用 Rekognition 主控台訓練轉接器時，您可以使用主控台選擇標籤，然後標記與標籤對應的映像來為映像新增註解。透過此程序，模型會學習如何根據指定的標籤識別映像的元素。此連結程序可讓模型在建立轉接器時專注於最相關的內容，從而提高映像分析的準確度。

或者，您可以提供資訊清單檔案，其中包含映像的相關資訊以及隨之附註解的資訊。

訓練和測試資料集

訓練資料集是微調模型和建立自訂配接器的基礎。您必須提供已註解的訓練資料集，以供模型學習。模型會從此資料集中學習，以改善其在您提供的映像類型上的效能。

為了提高準確性，您必須通過註釋/標記映像來建立訓練資料集。您可以透過兩種方式完成此操作：

- 手動指派標籤：您可以使用 Rekognition 主控台建立訓練資料集，方法是上傳您要資料集包含的映像，然後手動指派標籤給這些映像。
- 資訊清單檔案：您可使用資訊清單檔案來訓練轉接器。資訊清單檔案包含訓練和測試映像的基本真相註解，以及訓練映像位置的資訊。您可以在使用 Rekognition API 訓練介面卡時或使用主控台時提供資訊清單檔案。 AWS

測試資料集用於在訓練後評估轉接器的效能。為了確保可靠的評估，測試資料集是通過使用模型以前從未見過的原始培訓資料集的一部分來建立。此程序可確保使用新資料評估轉接器的效能，從而建立精確的測量和指標。若要取得最佳精確度的改進，請參閱 訓練轉接器的最佳實務。

使用 AWS CLI 和 SDK 管理介面卡

Rekognition 可讓您利用多種運用預先訓練的電腦視覺模型的功能。使用這些模型，您可以執行諸如標籤偵測和內容管制之類的任務。您也可以使用轉接器自訂這些特定型號。

您可以使用 Rekognition 的專案建立和專案管理 API (例如和CreateProject版本) 來建立CreateProject和訓練配接器。下列頁面說明如何使用 API 作業，使用 AWS 主控台、您選擇的 AWS SDK 或 AWS CLI 來建立、訓練和管理介面卡。

訓練轉接器之後，您可以在使用支援的功能執行推論時使用。目前，使用內容管制特徵時支援轉接器。

使用 AWS SDK 訓練適配器時，必須以清單文件的形式提供基本真相標籤（圖像註釋）。或者，您可使用 Rekognition 主控台來建立和訓練轉接器。

> **Note**
>
> 無法複製介面卡。只能複製 Rekognition 自訂標籤專案版本。

主題

介面卡狀態

自訂協調配接器 (專案版本) 可處於下列其中一種狀態：

- 訓練進度-轉接器正在針對您提供的訓練文件的檔案進行訓練。
- 訓練 _ 已完成-介面卡已成功完成訓練，並準備好讓您檢閱其效能。
- TRAINING_FAILED-介面卡因某些原因無法完成訓練，請檢閱輸出資訊清單檔案和輸出資訊清單摘要，以取得失敗原因的相關資訊。
- 刪除-轉接器正在刪除。
- 已取代-介面卡已在舊版的「內容協調」基本模型上進行訓練。這是在寬限期內，它將在新基礎模型版本發布後的 60 到 90 天內到期。在寬限期內，您仍然可以使用配接器進行DetectModeration標籤或 StartMediaAnalysisJobAPI 作業的推論。如需介面卡的到期日，請參閱自訂協調主控台。

- 已過期-介面卡已在較舊版本的「內容協調」基本模型上進行訓練，因此無法再用於透過 DetectModerationLabels 或 StartMediaAnalysisJob API 作業取得自訂結果。如果在推論請求中指定了「過期」配接卡，則會略過該配接卡，而是從最新版本的「自訂協調」基本模型傳回回應。

建立專案

透過此CreateProject作業，您可以建立一個專案，以保留 Rekognition 標籤偵測作業的介面卡。專案是一組資源，在標籤偵測作業 (例如) 的情況下 DetectModerationLabels，專案可讓您儲存可用來自訂基礎 Rekognition 模型的介面卡。呼叫時 CreateProject，您可以將要建立的專案名稱提供給 ProjectName 引數。

若要使用 AWS 主控台建立專案：

- 簽署 Rekognition 主控台
- 按一下自訂管制
- 選擇建立專案
- 選擇建立新專案或新增到現有專案
- 新增專案名稱
- 新增轉接器名稱
- 視需要新增描述
- 選擇您要匯入訓練映像的方式：資訊清單檔案、S3 儲存貯體或從電腦匯入
- 選擇是否要自動分割訓練資料或匯入資訊清單檔案
- 選擇是否要自動更新專案
- 按一下建立專案。

若要使用 AWS CLI 和 SDK 建立專案：

1. 如果您尚未這麼做，請安裝並設定 AWS CLI 和 AWS SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用以下程式碼來建立專案。

CLI

```
# Request
```

```
# Creating Content Moderation Project
aws rekognition create-project \
    --project-name "project-name" \
    --feature CONTENT_MODERATION \
    --auto-update ENABLED
    --profile profile-name
```

描述專案

您可以使用 DescribeProjectsAPI 取得專案的相關資訊，包括與專案相關聯之所有轉接器的相關資訊。

若要使用 AWS CLI 和 SDK 描述專案：

1. 如果您尚未這麼做，請安裝並設定 AWS CLI 和 AWS SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用下列程式碼來描述專案：

CLI

```
# Request
# Getting CONTENT_MODERATION project details
aws rekognition describe-projects \
    --features CONTENT_MODERATION
    --profile profile-name
```

刪除專案

您可以使用 Rekognition 主控台或呼叫 API 來刪除專案。DeleteProject若要刪除專案，您必須先刪除每一個相關轉接器。刪除的專案或模型無法取消刪除。

若要使用 AWS 主控台刪除專案：

- 簽署 Rekognition 主控台。
- 按一下自訂管制。
- 您必須刪除專案中的每一個轉接器，才能刪除專案本身。選取轉接器，然後選取刪除，以刪除與專案相關聯的所有轉接器。

- 選取專案，然後選取刪除按鈕。

若要使用 AWS CLI 和 SDK 刪除專案：

1. 如果您尚未這麼做，請安裝並設定 AWS CLI 和 AWS SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用下列程序來刪除專案。

CLI

```
aws rekognition delete-project
  --project-arn project_arn \
  --profile profile-name
```

建立專案版本

您可以使用「CreateProject版本」作業來訓練用於部署的介面卡。 CreateProjectVersion 首先建立與專案關聯的轉接器的新版本，然後開始訓練轉接器。來自的響應 CreateProjectVersion 是模型版本的 Amazon 資源名稱（ARN）。培訓需要一段時間才能完成。您可以通過調用獲取當前狀態 DescribeProjectVersions。訓練模型時，Rekognition 會使用與專案相關聯的訓練和測試資料集。可以使用主控台來建立資料集。如需詳細資訊，請參閱頁面上的資料集。

若要使用 Rekognition 主控台建立專案版本：

- 簽署 AWS Rekognition 主控台
- 按一下自訂管制
- 選取專案。
- 在「專案詳細資訊」頁面上，選擇建立轉接器
- 在「建立專案」頁面上，填入專案詳細資料、訓練映像和測試映像的必要詳細資料，然後選取建立專案。
- 在「為映像指派標籤」頁面上，為映像新增標籤，完成後選取開始訓練

若要使用 AWS CLI 和 SDK 建立專案版本：

1. 如果您尚未這麼做，請安裝並設定 AWS CLI 和 AWS SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用下面的程式碼來建立一個專案版本：

CLI

```
# Request
aws rekognition create-project-version \
 --project-arn project-arn \
 --training-data '{Assets=[GroundTruthManifest={S3Object="my-
bucket",Name="manifest.json"}]}' \
 --output-config S3Bucket=my-output-bucket,S3KeyPrefix=my-results \
 --feature-config "ContentModeration={ConfidenceThreshold=70}"
 --profile profile-name
```

描述專案版本

您可以使用「DescribeProject版本」作業列出並描述與專案相關聯的轉接器。您最多可以在中指定 10 個模型版本 ProjectVersionArns。如果您不指定值，就會傳回專案中所有模型版本的描述。

若要使用 AWS CLI 和 SDK 描述專案版本：

1. 如果您尚未這麼做，請安裝並設定 AWS CLI 和 AWS SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 請使用下列程式碼來描述專案版本：

CLI

```
aws rekognition describe-project-versions
  --project-arn project_arn \
  --version-names [versions]
```

刪除專案版本

您可以使用「DeleteProject版本」作業刪除與專案相關聯的 Rekognition 轉接器。如果轉接器正在執行或正在訓練，您就無法刪除轉接器。若要檢查介面卡的狀態，請呼叫 DescribeProjectVersions 作業並檢查其傳回的「狀態」欄位。停止執行中的介面卡呼叫 StopProjectVersion。如果模型正在訓練，請等到完成訓練以刪除模型。您必須刪除專案中的每一個轉接器，才能刪除專案本身。

若要使用 Rekognition 主控台刪除專案版本：

- 簽署 Rekognition 主控台
- 按一下自訂管制
- 從專案標籤中，您可以看到所有專案和相關聯的轉接器。選取轉接器，然後選取刪除。

若要使用 AWS CLI 和 SDK 刪除專案版本：

1. 如果您尚未這麼做，請安裝並設定 AWS CLI 和 AWS SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和開 AWS 發套件。
2. 使用下列程式碼來刪除專案版本：

CLI

```
# Request
aws rekognition delete-project-version
  --project-version-arn model_arn \
  --profile profile-name
```

自訂管制轉接器教學課程

本教學課程說明如何使用 Rekognition 主控台建立、訓練、評估、使用和管理轉接器。若要使用 AWS SDK 建立、使用和管理配接器，請參閱使用 AWS CLI 和 SDK 管理介面卡。

配接器可讓您增強 Rekognition API 操作的準確性，自訂模型的行為以符合自身需求和使用案例。在您使用本教學課程建立介面卡之後，您可以在使用 Label 等DetectModeration操作分析自己的影像時使用它，以及重新訓練介面卡以進一步 future 進。

於本教學課程中，您會了解：

- 使用 Rekognition Console 建立專案
- 為訓練資料加上註解
- 在訓練資料集上訓練您的轉接器
- 檢閱轉接器的效能
- 使用您的轉接器進行映像分析

必要條件

在完成本教學課程之前，建議您先閱讀 建立和使用轉接器。

若要建立轉接器，您可以使用 Rekognition 主控台工具來建立專案、上傳和註解您自有映像，然後在這些映像上訓練配接器。若要開始使用，請參閱 建立專案並訓練轉接器。

或者，您可以使用 Rekognition 的主控台或 API 擷取映像的預測，然後在針對這些預測訓練轉接器之前驗證預測。若要開始使用，請參閱 批量分析、預測驗證和訓練轉接器。

映像註釋

您可以使用 Rekognition 主控台為映像加上標籤，或使用 Rekognition 批量分析來註解映像，然後可以驗證這些映像已正確標記。選擇以下其中一個主題以開始使用。

主題

- 建立專案並訓練轉接器
- 批量分析、預測驗證和訓練轉接器

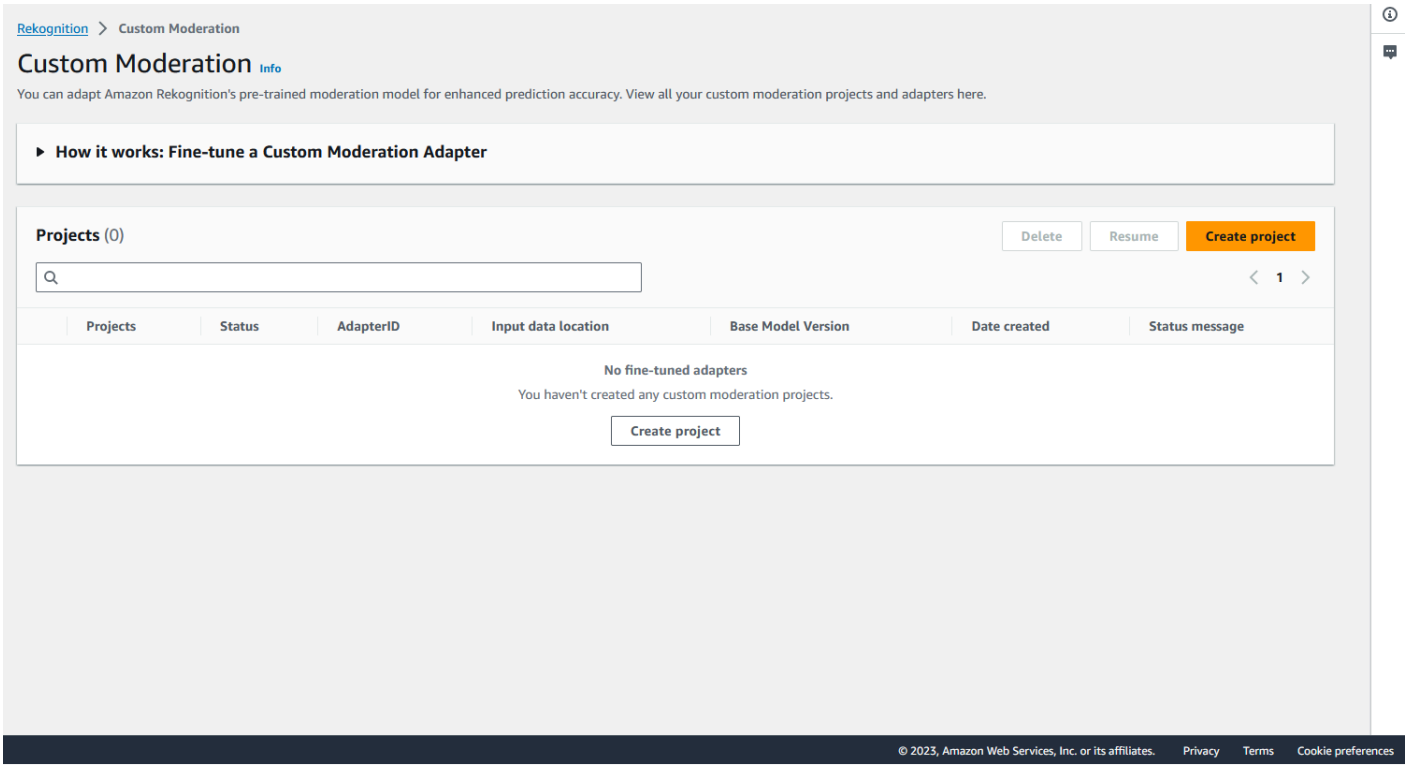
建立專案並訓練轉接器

使用 Rekognition 主控台註釋映像，完成下列步驟以訓練轉接器。

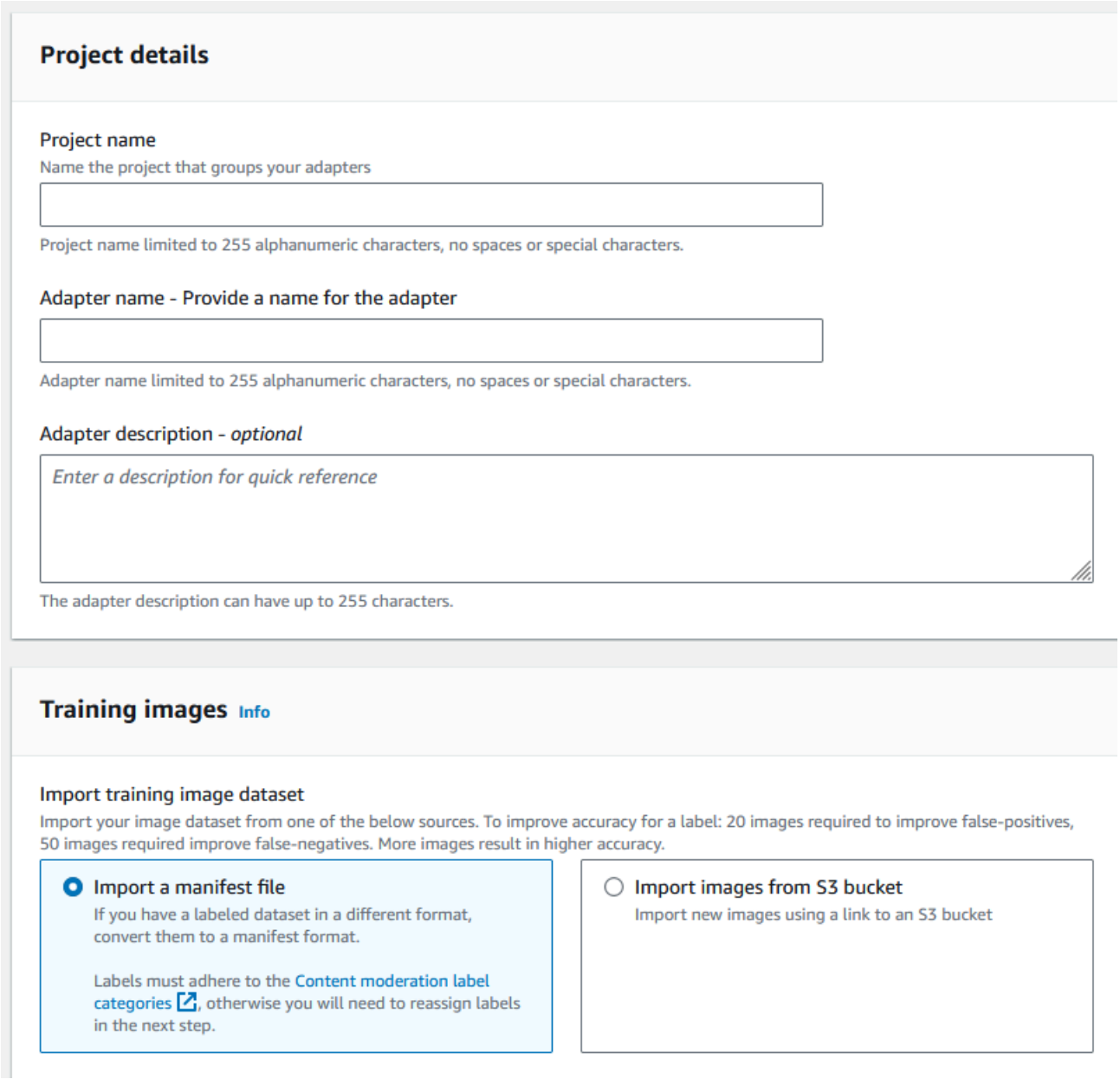
建立專案

在訓練或使用轉接器之前，您必須先建立將包含轉接器的專案。您也必須提供轉接器訓練轉接器時使用的映像。若要建立專案、轉接器和映像資料集：

1. 登入 AWS 管理主控台，然後開啟 Rekognition 主控台，網址為 https://console.aws.amazon.com/rekognition/。
2. 在左窗格中，選擇自訂管制。此時會顯示 Rekognition 自訂管制的登陸頁面。



3. 自訂管制登陸頁面會顯示所有專案和轉接器的清單，而且還有一個按鈕可用來建立轉接器。選擇建立專案以建立新的專案和轉接器。
4. 如果這是您第一次建立轉接器，系統會提示您建立 Amazon S3 儲存貯體來存放與專案和轉接器相關的檔案。選擇建立 Amazon S3 儲存貯體。
5. 在下一頁中，填寫轉接器名稱和專案名稱。如果需要，請提供轉接器說明。

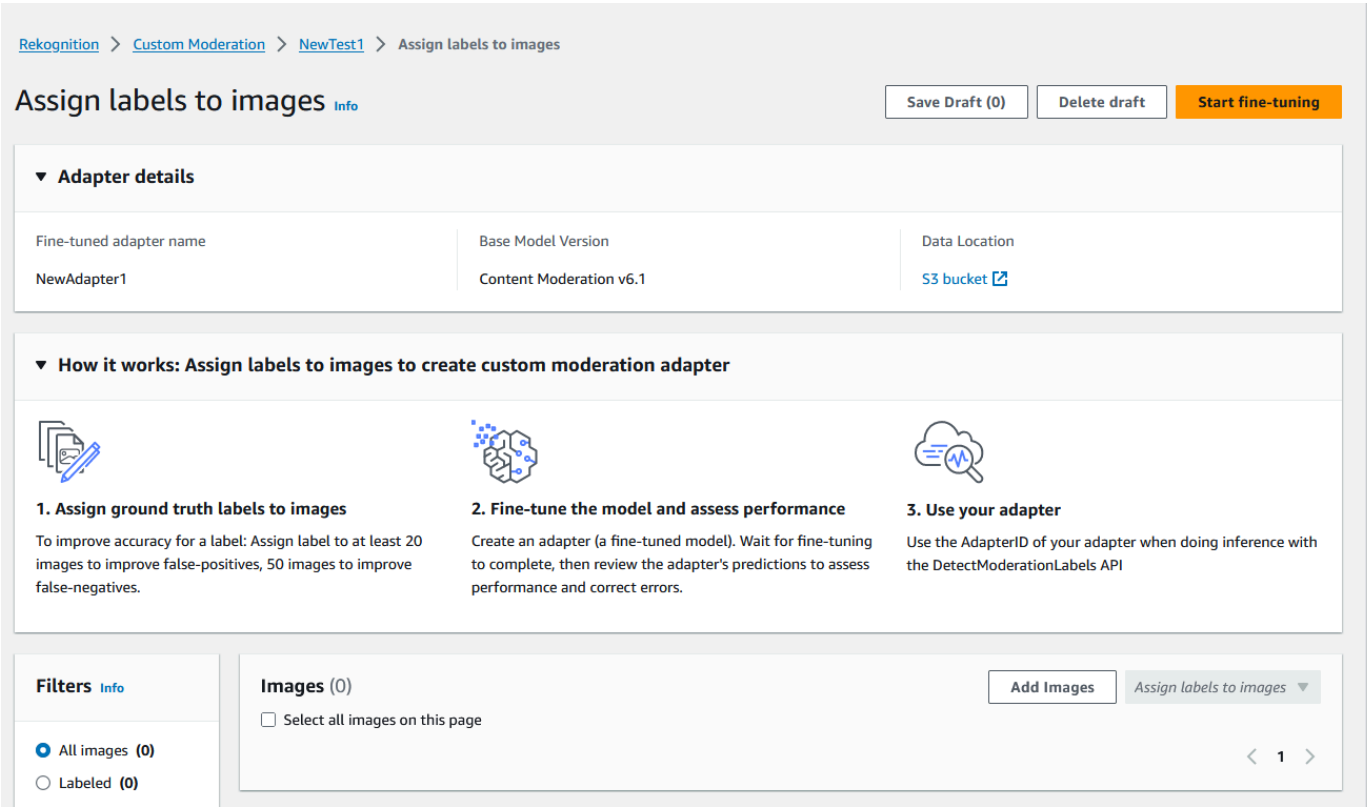


6. 在此步驟中，您也會提供轉接器的映像。您可以選取：從電腦匯入映像、匯入資訊清單檔案或從 Amazon S3 儲存貯體匯入映像。如果您選擇從 Amazon S3 儲存貯體匯入映像，請提供包含訓練映像之儲存貯體和資料夾的路徑。如果您直接從電腦上傳映像，請注意，一次最多只能上傳 30 張映像。如果您使用的是包含註釋的資訊清單檔案，您可以略過下列涵蓋映像註釋的步驟，然後繼續執行 檢視轉接器效能 上的區段。
7. 在測試資料集詳情區段中，選擇自動分割，讓 Rekognition 自動選取適當的映像百分比做為測試資料，或者也可以選擇手動匯入資訊清單檔案。
8. 填寫此資訊後，選取建立專案。

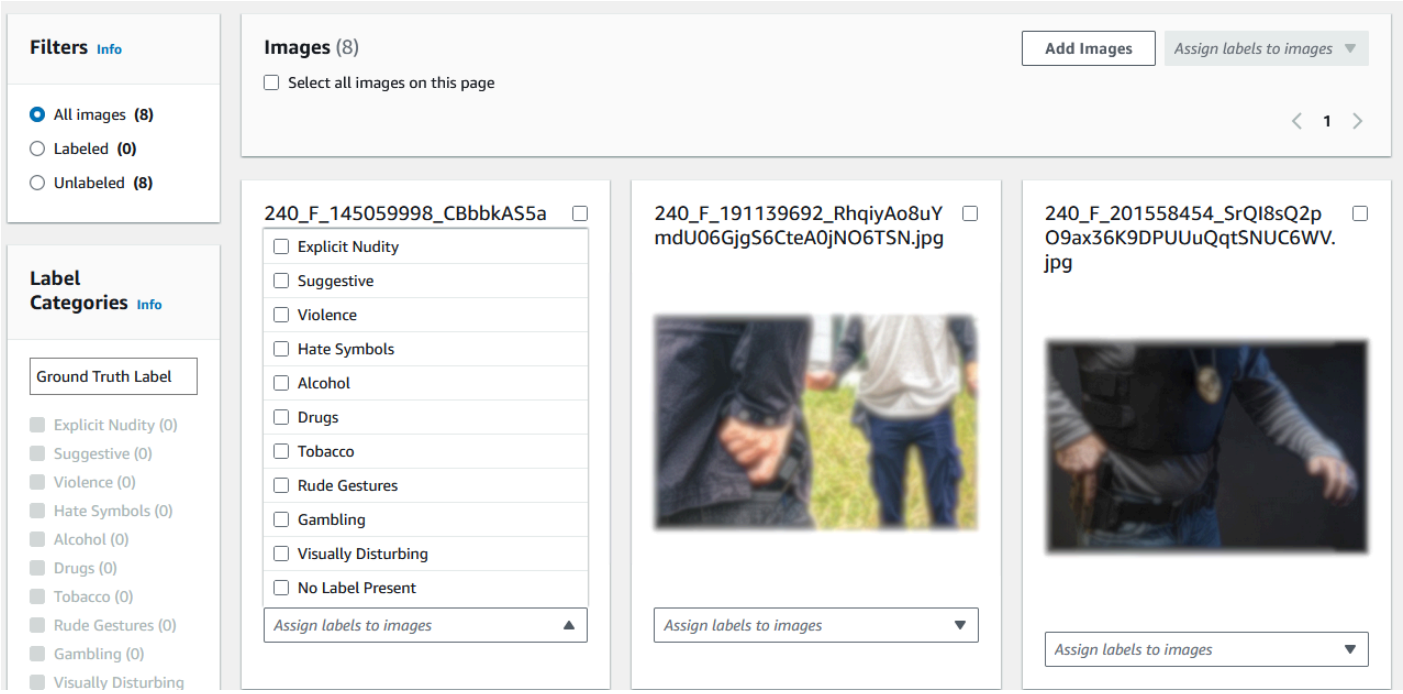
訓練轉接器

要在自己未註釋的映像上訓練轉接器：

1. 選取包含轉接器的專案，然後選擇指派標籤給映像的選項。
2. 在指派標籤給映像的頁面上，可以看到已上傳為訓練映像的所有映像。您可以使用左側的兩個屬性選取面板，依標籤/未標示狀態和標籤類別篩選這些映像。您可以選取新增映像按鈕，將其他映像新增至訓練資料集。



3. 將映像新增至訓練資料集後，您必須使用標籤為映像加上註解。上傳映像後，「為映像分配標籤」頁面將更新以顯示您上傳的映像。系統將提示您從 Rekognition 管制支援的標籤下拉式清單中選取適合您映像的標籤。您可選擇多個專案。
4. 繼續此程序，直到您已將標籤新增至訓練資料中的每個映像。
5. 標記所有資料之後，請選取開始訓練開始訓練模型，以建立您的轉接器。



6. 在開始訓練程序之前，您可以將任何標籤新增至您想要的轉接器。您也可以為介面卡提供自訂加密金鑰或使用 AWS KMS 金鑰。完成新增任何想要的標籤並根據您的喜好自訂加密之後，請選取訓練轉接器以開始轉接器的訓練程序。
7. 等待轉接器完成訓練。訓練完成後，您會收到轉接器已完成建立的通知。

轉接器的狀態為「訓練完成」之後，您就可以檢視轉接器的指標

批量分析、預測驗證和訓練轉接器

透過驗證 Rekognition 的內容管制模型中的批量分析預測，完成以下步驟來訓練您的轉接器。

若要透過驗證 Rekognition 的內容管制模型中的預測來訓練轉接器，您必須：

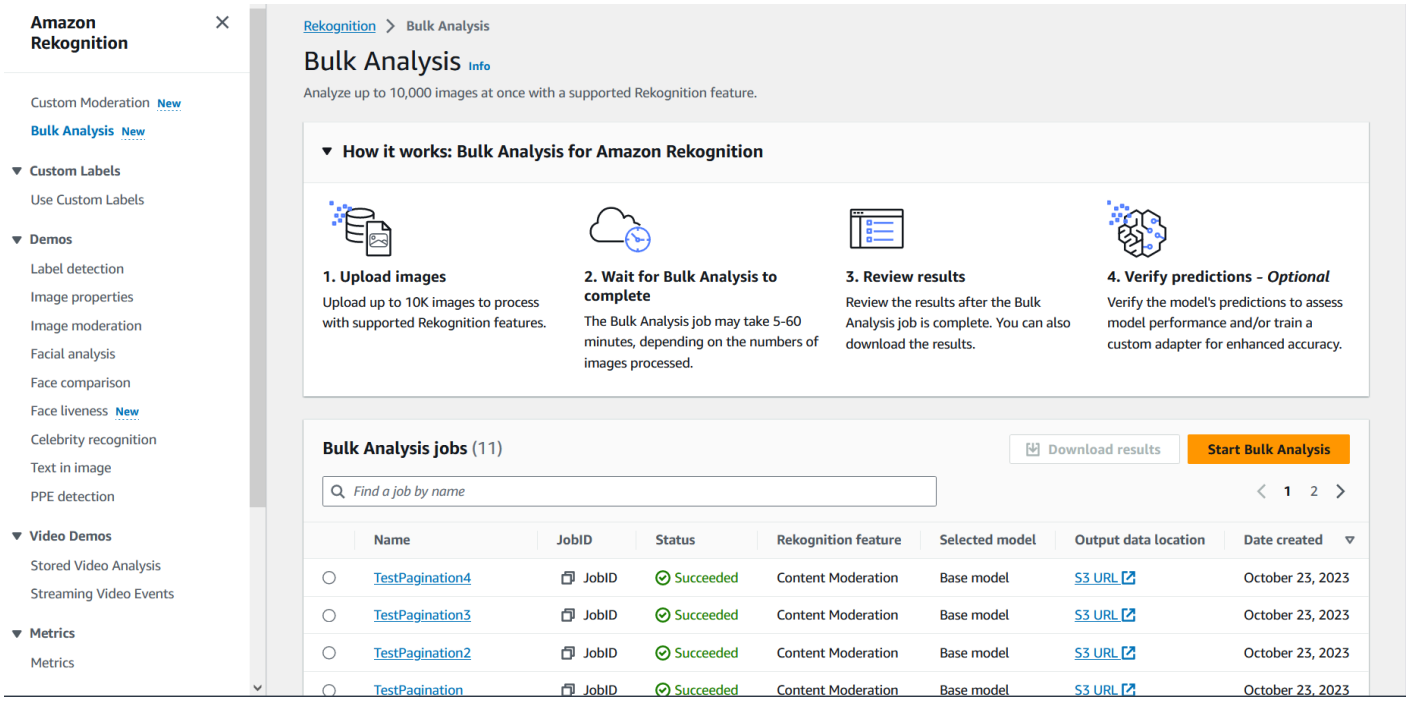
1. 對映像進行批量分析
2. 驗證為您的映像傳回的預測

您可以使用 Rekognition 的基礎模型或您已建立的轉接器執行批量分析，以取得映像預測。

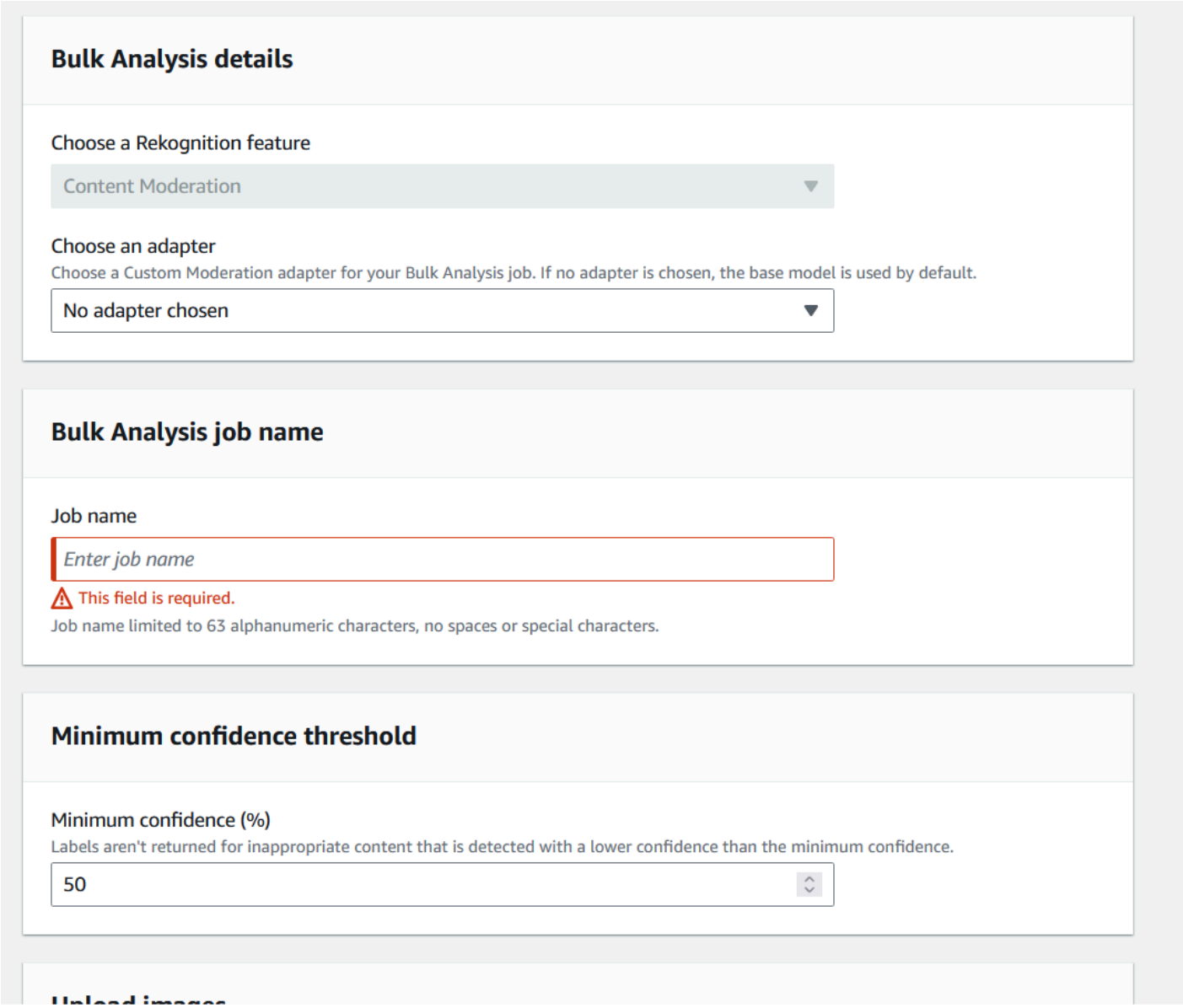
對映像執行批量分析

若要針對已驗證的預測訓練轉接器，您必須先啟動批量分析工作，使用 Rekognition 的基本模型或您選擇的轉接器來分析一批映像。若要執行批量分析操作：

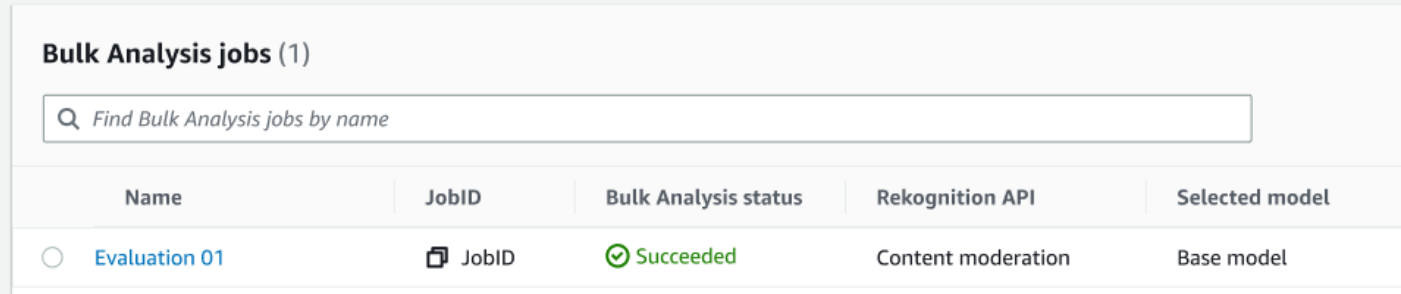
1. 登入 AWS Management Console 並開啟亞馬遜重新認知主控台，網址為 https://console.aws.amazon.com/rekognition/。
2. 在左窗格中，選擇批量分析。批量分析登陸頁面隨即出現。選擇開始批量分析。「批量分析」功能概觀顯示上傳影像、等待分析、檢閱結果以及選擇性地驗證模型預測的步驟。使用基礎模型列出內容協調的最近大量分析工作。



3. 如果這是您第一次建立轉接器，系統會提示您建立 Amazon Simple Storage Service 儲存貯體，以存放與您的專案和轉接器相關的檔案。選擇建立 Amazon S3 儲存貯體。
4. 使用選擇轉接器下拉式功能表，選取要用於批量分析的轉接器。如果未選擇轉接器，則預設會使用基礎模型。基於本教學的用途，請選擇轉接器。



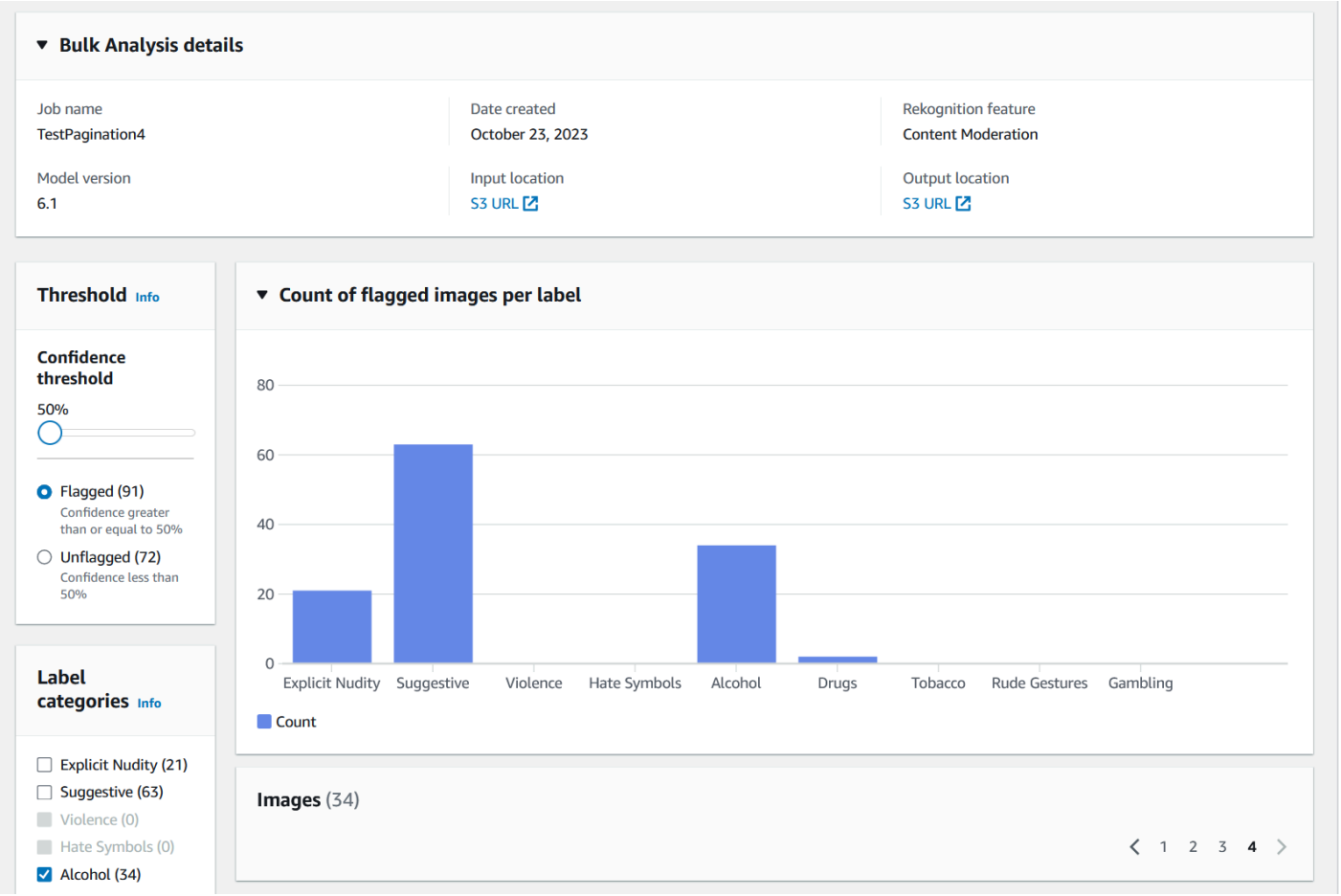
5. 在批量分析作業名稱欄位中，填入批量分析作業名稱。
6. 選擇最小可信度閾值的值。不會傳回低於您選擇的信賴閾值的標籤預測。請注意，稍後評估模型的效能時，將無法調整置信度閾值低於所選的最低可信度閾值。
7. 在此步驟中，您還將使用批量分析提供要分析的映像。這些映像也可以用來訓練轉接器。您可以選擇從電腦上傳映像或從 Amazon S3 儲存貯體匯入映像。如果您選擇從 Amazon S3 儲存貯體匯入文件，請提供包含訓練映像檔的儲存貯體和資料夾的路徑。如果您直接從電腦上傳文件，請注意，一次只能上傳 50 張映像。
8. 填寫此資訊後，選擇開始分析。這將使用 Rekognition 的基礎模型開始分析過程。
9. 您可以在批量分析主頁面上檢查作業的批量分析狀態，以檢查批量分析作業的狀態。當批量分析狀態變為「成功」時，即可檢閱分析結果。



10.從批量分析作業清單中選擇您建立的分析。

11.在批量分析詳細資料頁面上，您可以看到 Rekognition 基本模型針對您上傳的映像所做的預測。

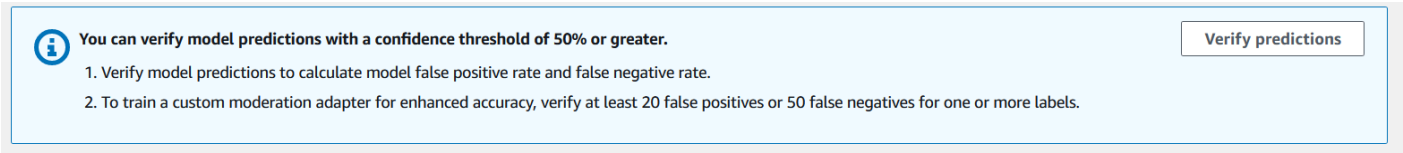
12.檢閱基礎模型的效能。您可以使用可信度閾值滑桿，變更轉接器必須具備的可信度閾值，才能將標籤指派給映像。當您調整可信度閾值時，已標記和未標記的執行個體數目會隨之變更。標籤類別的面板會顯示 Rekognition 可辨識的最上層類別，您可以在此清單中選取一個類別，以顯示已指派該標籤的任何映像。



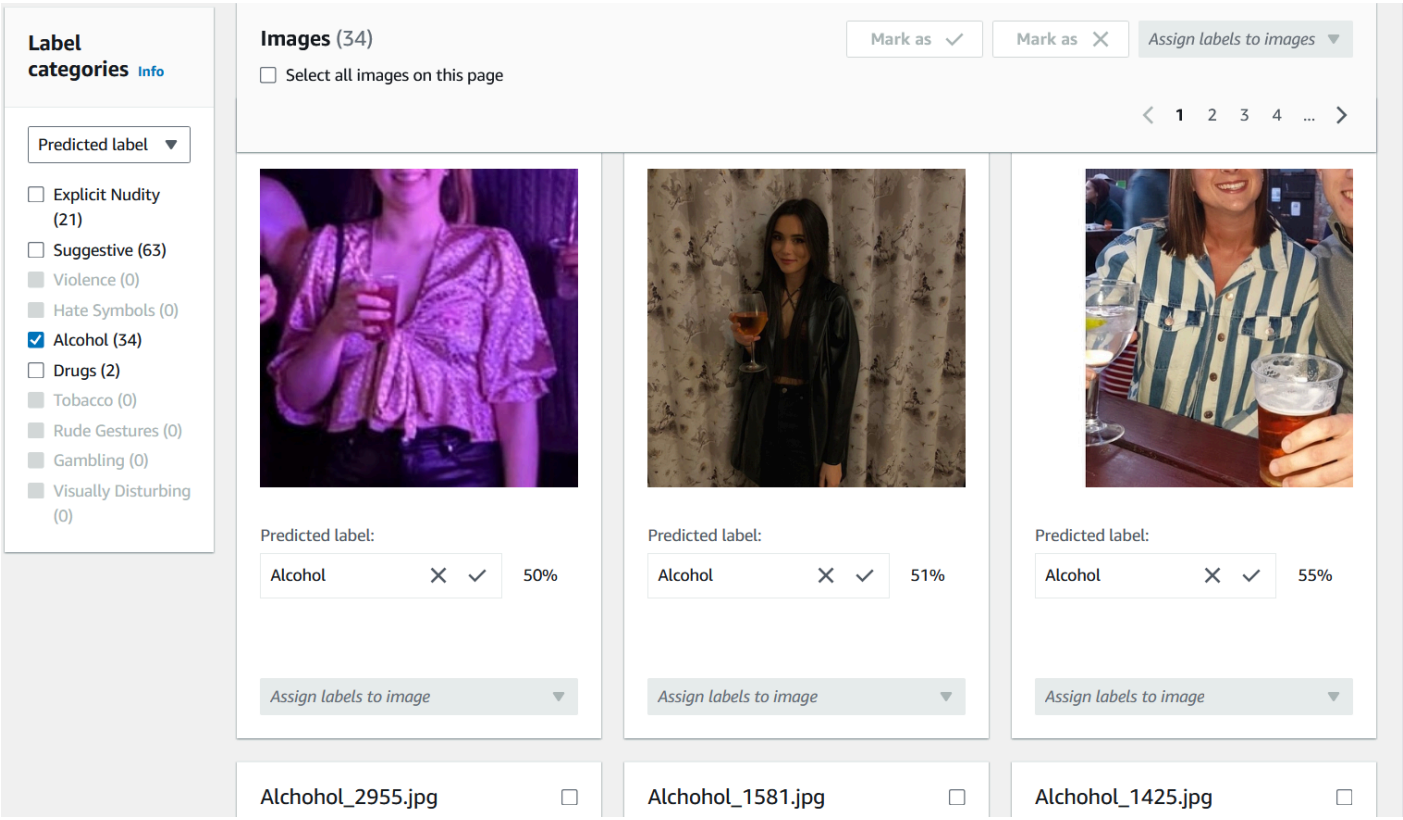
驗證預測

如果您已檢閱 Rekognition 基礎模型或所選轉接器的準確性，並且想要改善此精確度，可以利用驗證工作串流程：

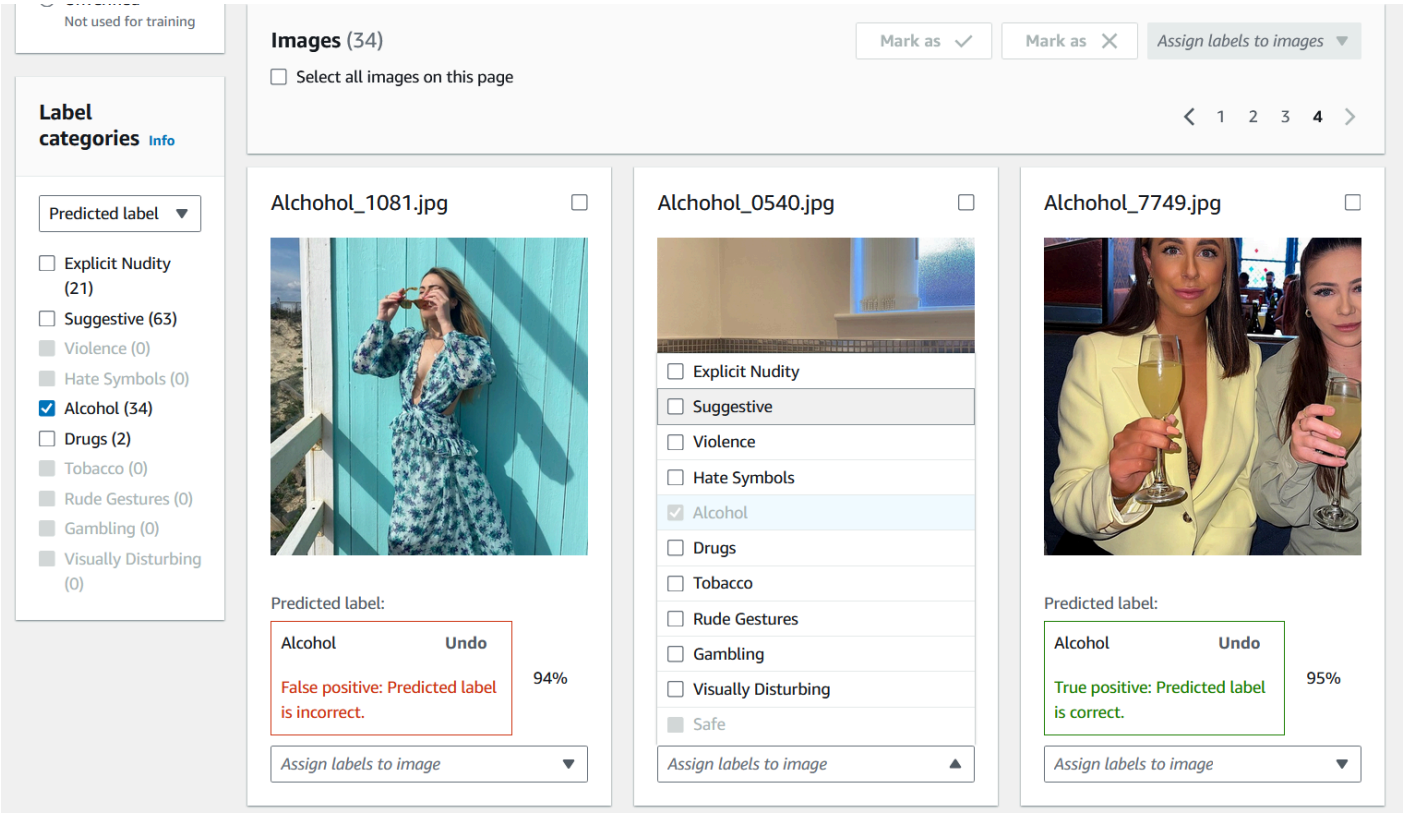
1. 完成基礎模型效能的檢閱後，您將需要驗證預測。更正預測支援您訓練轉接器。從批量分析頁面頂端選擇驗證預測。



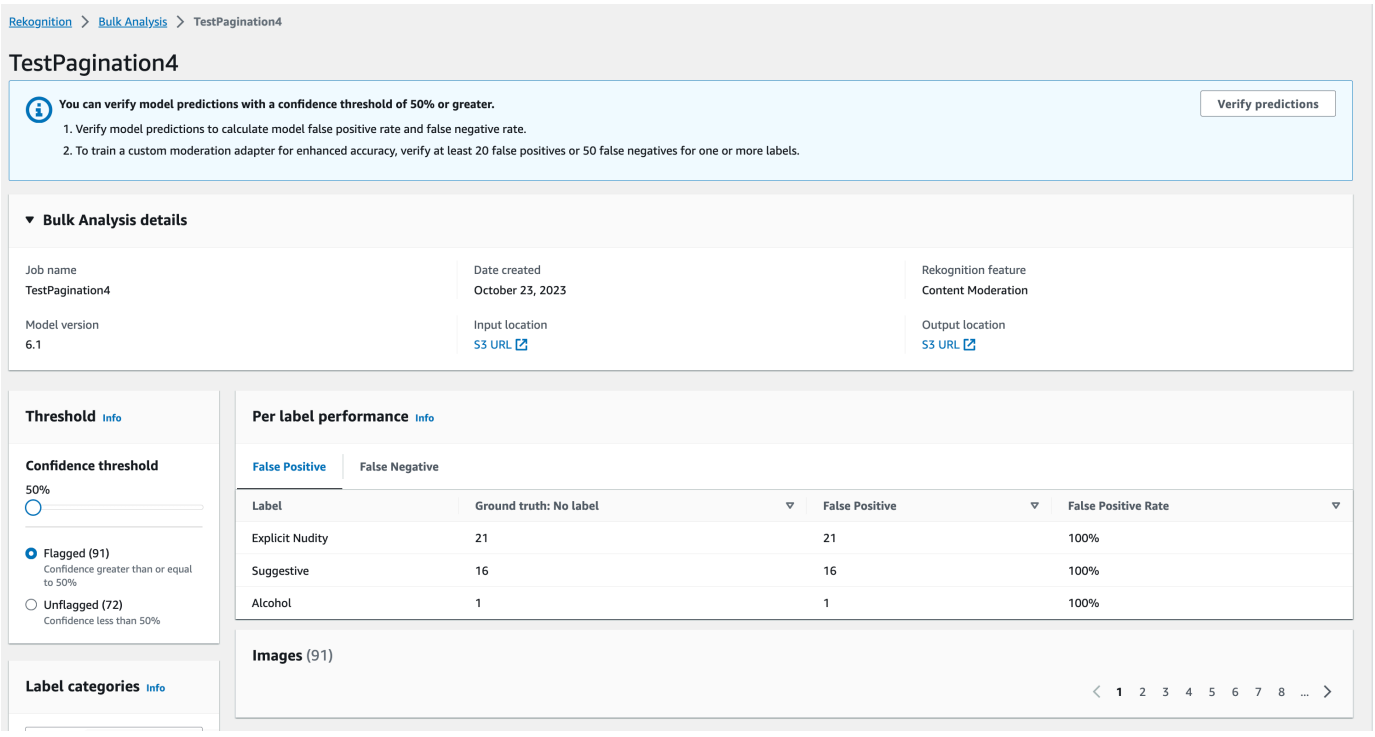
2. 在驗證預測頁面上，可以看到您提供給 Rekognition 基礎模型或所選轉接器的所有映像，以及每個映像的預測標籤。您必須使用映像下方的按鈕驗證每個預測是否正確或不正確。使用「X」按鈕將預測標記為不正確，使用複選標記按鈕將預測標記為正確。若要訓練轉接器，您需要驗證至少 20 個假陽性預測，以及指定標籤的 50 個假陰性預測。您驗證的預測越多，轉接器的效能就越好。



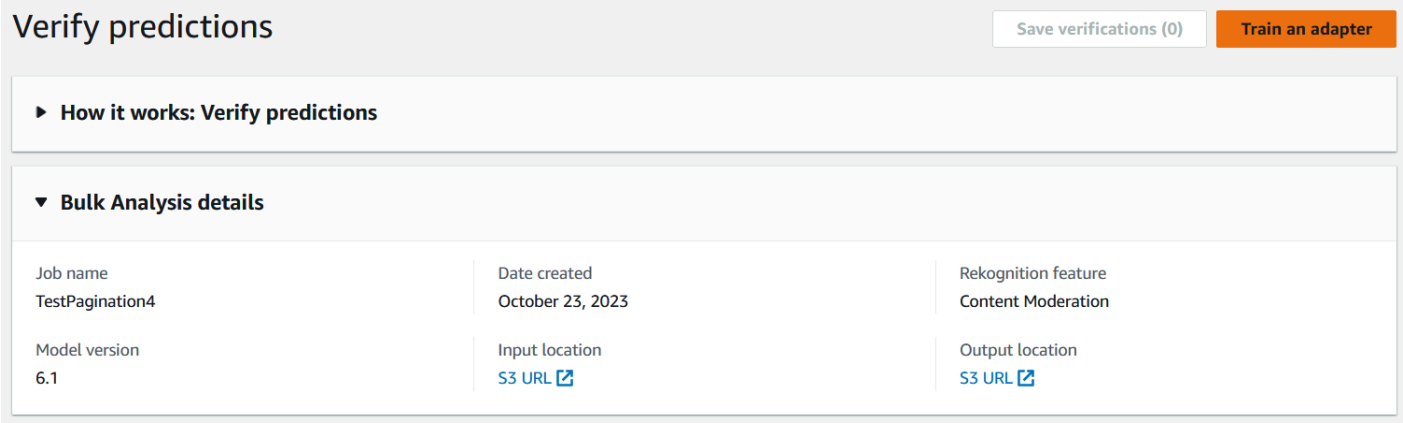
驗證預測後，映像下方的文字將會變更，以顯示您已驗證的預測類型。驗證映像後，您也可以使用將標籤指派給映像選單，為映像新增其他標籤。您可以查看模型針對您選擇的置信度閾值標記或未標記哪些映像，或依類別篩選映像。



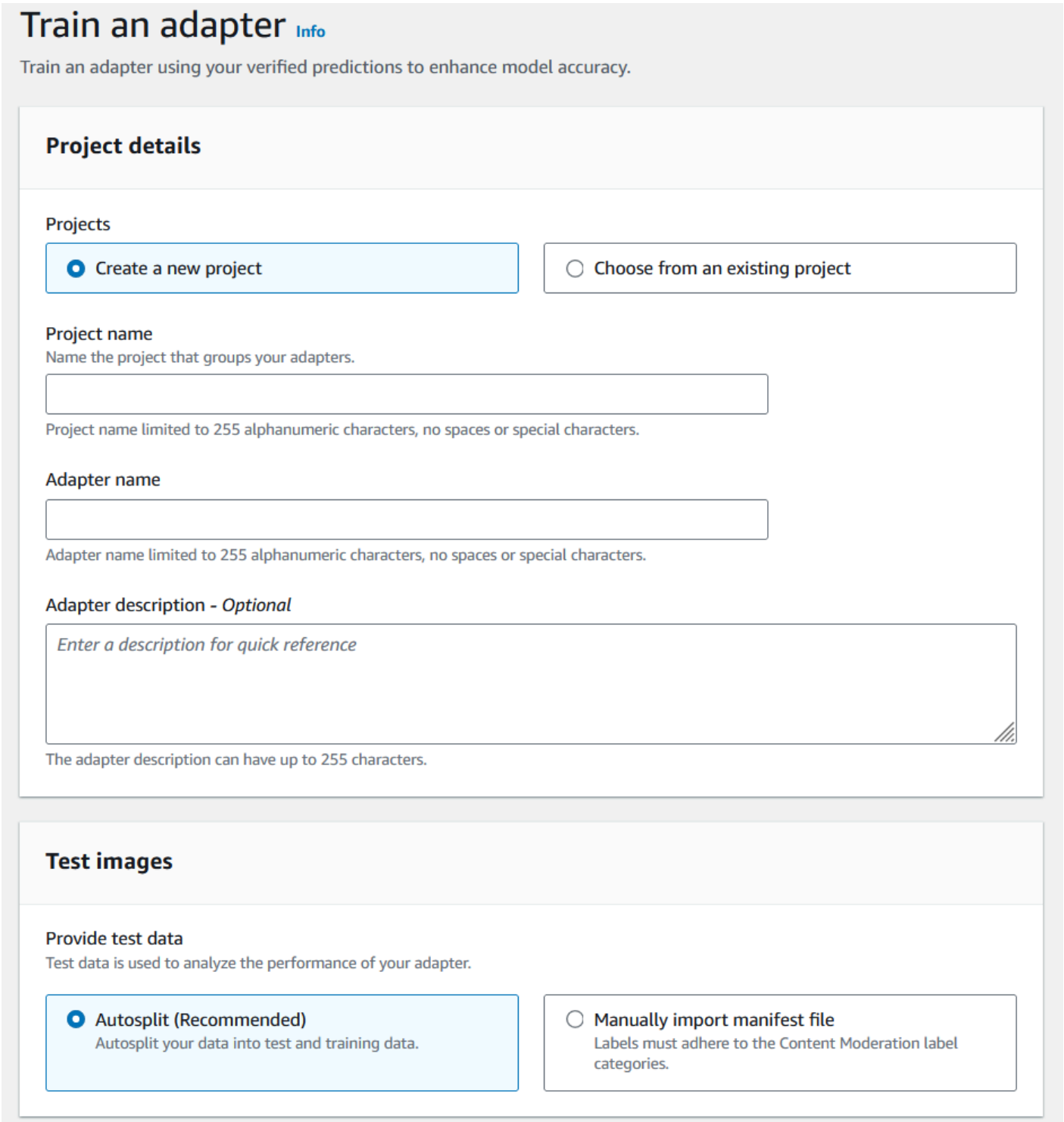
3. 完成驗證所有要驗證的預測後，您可以在驗證頁面的依標籤效能部分查看有關已驗證預測的統計資料。您也可以傳回批量分析詳細資訊頁面來檢視這些統計資料。



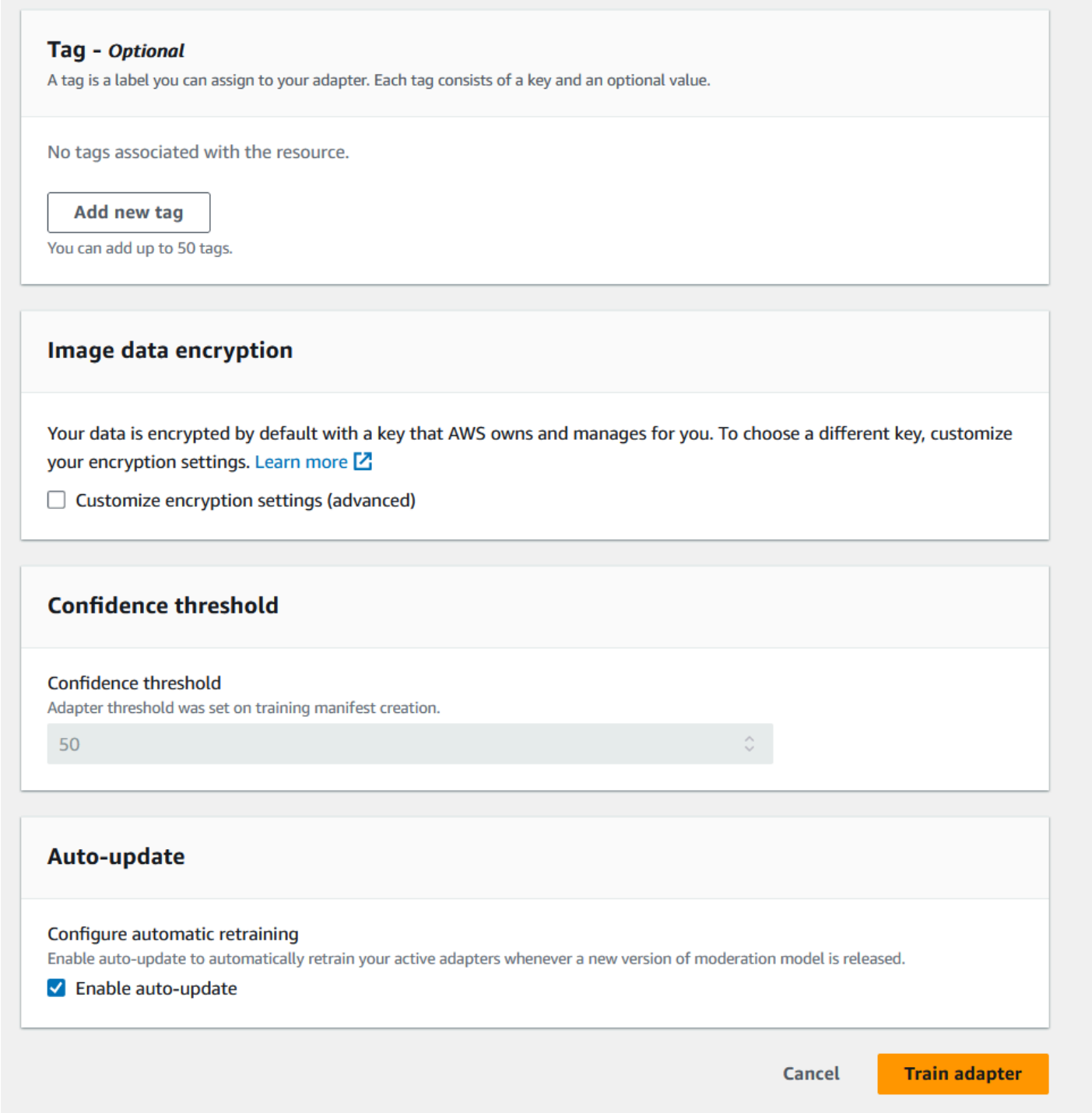
4. 如果您對依標籤效能的統計資料滿意，請再次前往驗證預測頁面，然後選取訓練轉接器按鈕，開始訓練您的轉接器。



5. 在訓練轉接器頁面上，系統會提示您建立專案或選擇現有的專案。命名專案和專案將包含的轉接器。您也必須指定測試映像的來源。指定映像時，您可以選擇自動分割讓 Rekognition 自動使用訓練資料的一部分作為測試映像，或者您也可以手動指定資訊清單檔案。建議選擇自動拆分。



6. 如果您不想使用預設 AWS KMS 金鑰，請指定您想要的任何標籤，以及 AWS 金鑰。建議將自動更新保留為啟用狀態。
7. 選擇培訓連接器。

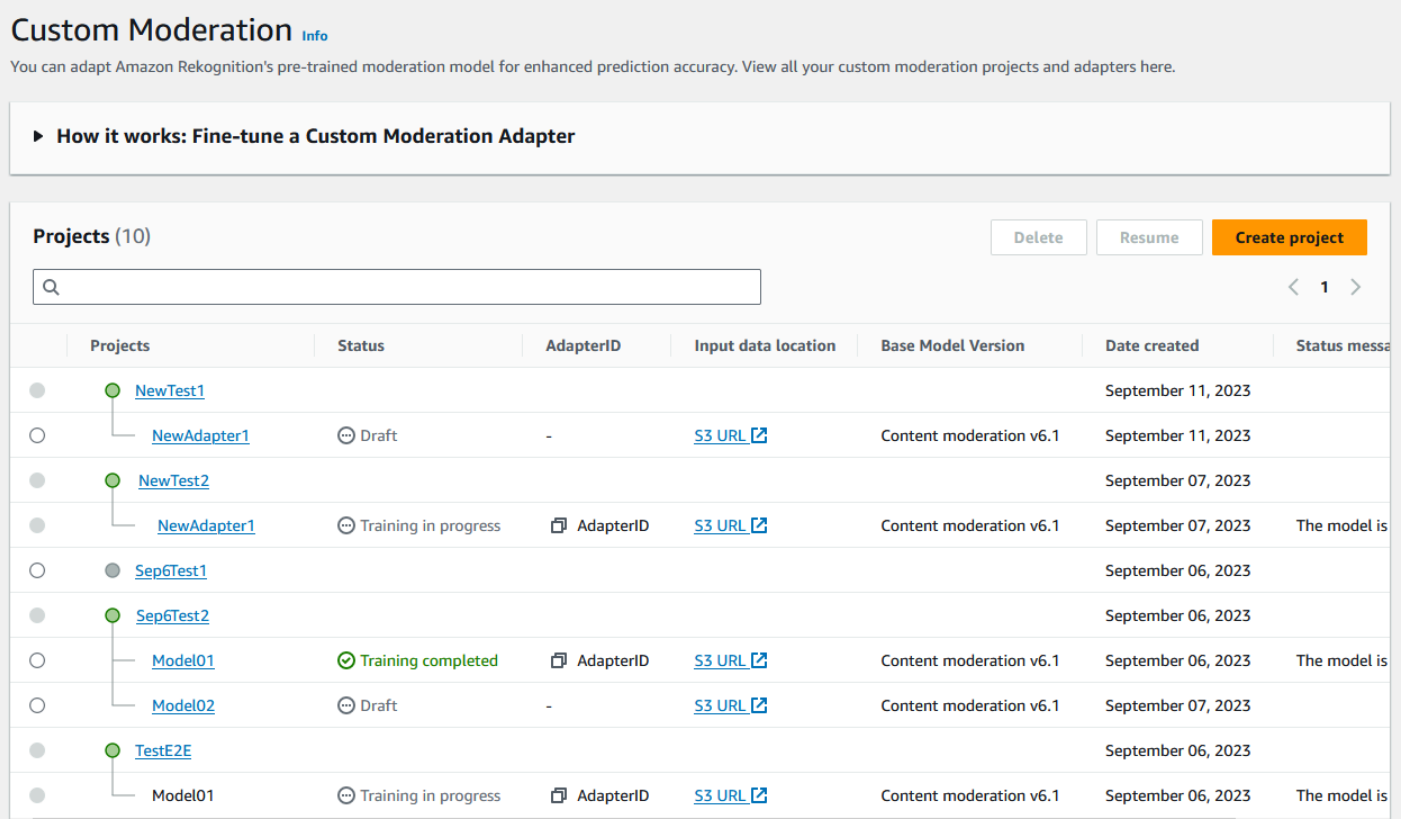


8. 一旦「自訂管制」登陸頁面上的轉接器狀態變成「訓練完成」，就可以檢視轉接器的效能。如需詳細資訊，請參閱檢視轉接器效能。

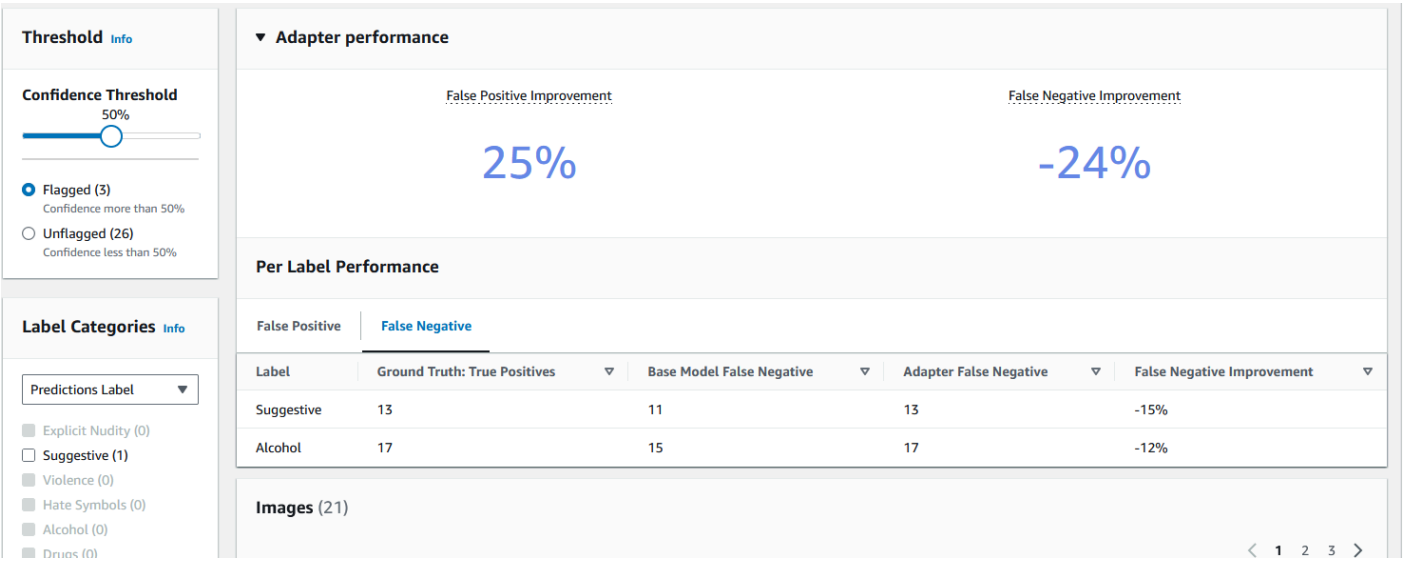
檢視轉接器效能

為檢視轉接器效能：

1. 使用主控台時，您可以在自訂管制登陸頁面的專案索引標籤下，查看與專案相關聯之任何轉接器的狀態。導覽至自訂管制登陸頁面。



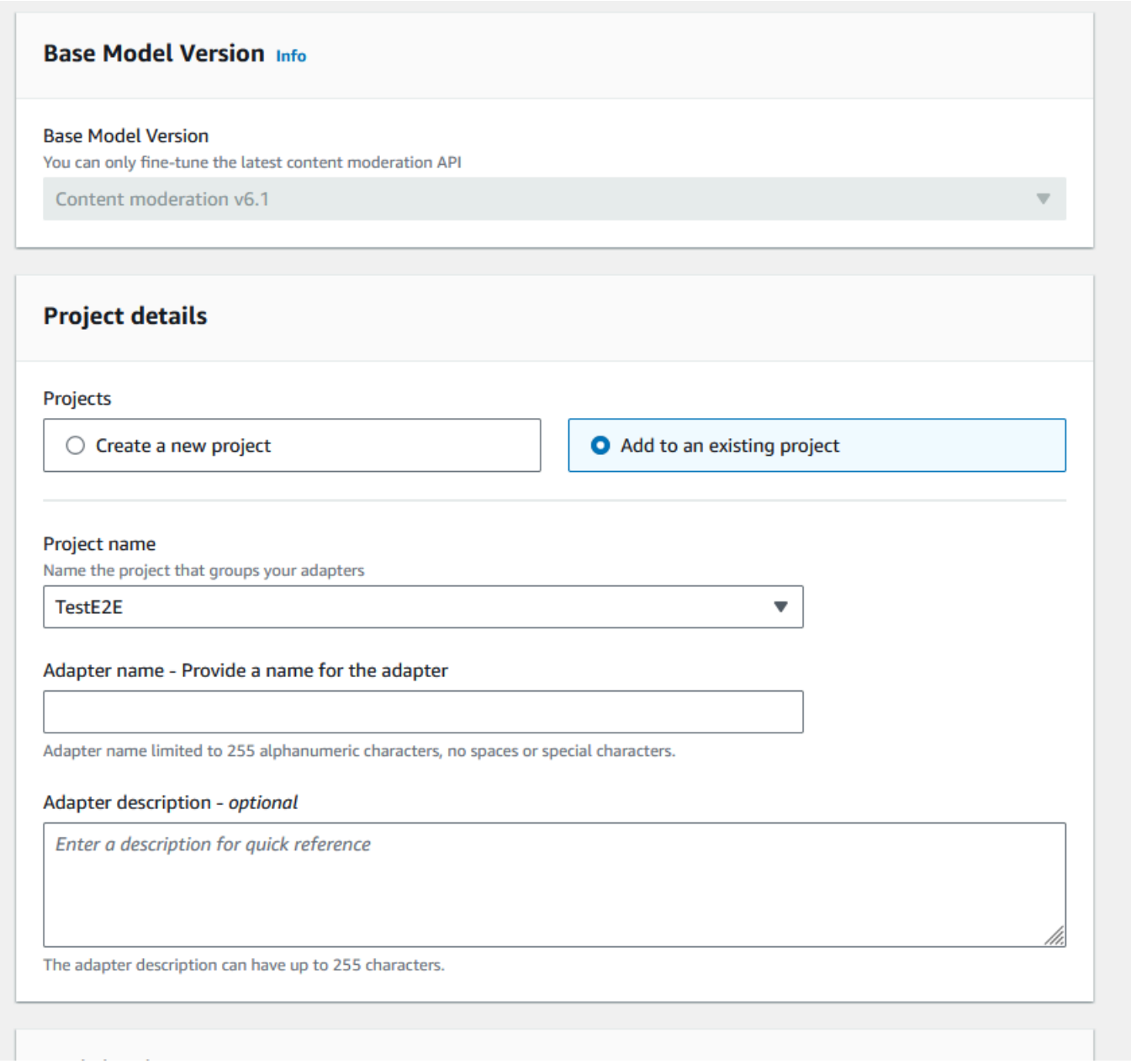
2. 從此清單中選取您要檢閱的轉接器。在下列轉接器詳細資料頁面上，您可以看到轉接器的各種度量。



3. 透過閾值面板，您可以變更轉接器為映像指派標籤所必須具備的最低可信度閾值。當您調整可信度閾值時，已標記和未標記的執行個體數目會隨之變更。您也可以依標籤類別篩選，以查看所選類別的指標。設定您選擇的閾值。
4. 您可以檢查轉接器效能面板中的指標，以評估轉接器在測試資料上的效能。這些指標的計算方式是將轉接器的擷取與測試集上的「基本真相」註解進行比較。

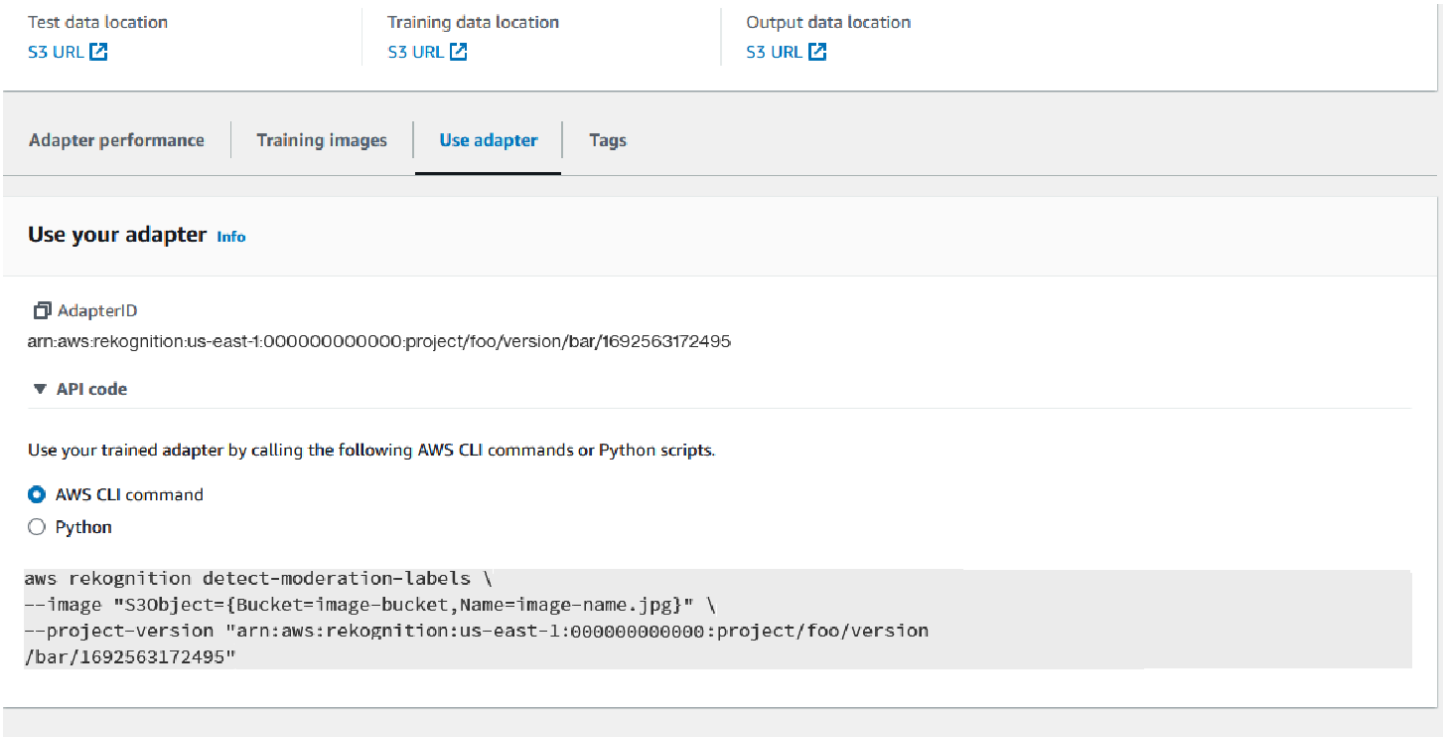
轉接器效能面板會顯示您建立之轉接器的偽陽性改善率和偽陰性改善率。按標籤效能標籤可用來比較轉接器和基礎模型在每個標籤類別上的效能。該標籤顯示了基本模型和轉接器的偽陽性和偽陰性預測的計數，按標籤類別分層。透過檢閱這些指標，您可以判斷轉接器需要改進的位置。如需這些指標的詳細資訊，請參閱 評估和改善您的轉接器。

若要改善效能，您可以收集更多訓練映像，然後在專案內部建立新的轉接器。只要傳回自訂管制登陸頁面，並在專案內建立新的轉接器，即可提供更多訓練映像供訓練的轉接器使用。這次選擇新增到現有專案選項，而不是建立新專案，然後從專案名稱下拉菜單中選擇要在其中建立新轉接器的專案。和以前一樣，為您的映像新增註釋或提供帶有註釋的資訊清單檔案。



使用轉接器

建立介面卡之後，您可以將其提供給支援的 Rekognition 作業，例如「標籤」。DetectModeration若要查看可用來使用介面卡執行推論的程式碼範例，請選取「使用介面卡」索引標籤，您可以在其中查看 AWS CLI 和 Python 的程式碼範例。您也可以造訪建立轉接器之操作文件的相應章節，以查看更多程式碼範例、設定指示和範例 JSON。



刪除您的轉接器和專案

您可以刪除個別轉接器，或刪除專案。您必須刪除專案中的每一個轉接器，才能刪除專案本身。

1. 若要刪除與專案相關聯的轉接器，請選擇轉接器，然後選擇刪除。
2. 若要刪除專案，請選擇您要刪除的專案，然後選擇刪除。

評估和改善您的轉接器

在每一輪轉接器訓練之後，您會想要檢閱 Rekognition 主控台工具中的效能指標，以判斷轉接器與您想要的效能等級有多接近。然後，您可以通過上傳一批新的訓練映像並在專案中訓練新的轉接器，來進一步提高轉接器的映像準確性。建立改良版的轉接器之後，您可以使用主控台刪除不再需要的任何舊版轉接器。

您也可以使用DescribeProject版本 API 作業擷取指標。

效能指標

完成訓練程序並建立轉接器之後，評估轉接器從映像擷取資訊的程度非常重要。

Rekognition 主控台提供了兩個指標，可協助您分析轉接器的效能：偽陽性改善和偽陰性改善。

您可以選取主控台轉接器部分的「轉接器效能」標籤，來檢視任何轉接器的這些指標結果。轉接器效能面板會顯示您建立之轉接器的偽陽性改善率和偽陰性改善率。

偽陰性改善措施會衡量轉接器對偽陰性的辨識，在基礎模型上有多少改善。如果偽陰性改善值為 25%，表示轉接器在測試資料集上將偽陰性的辨識提升 25%。

偽陰性改善措施會衡量轉接器對偽陰性的辨識，在基礎模型上有多少改善。如果偽陰性改善值為 25%，表示轉接器在測試資料集上將其對偽陰性的辨識提高 25%。

按標籤效能標籤可用來比較轉接器和基礎模型在每個標籤類別上的效能。該標籤顯示了基本模型和轉接器的偽陽性和偽陰性預測的計數，按標籤類別分層。透過檢閱這些指標，您可以判斷轉接器需要改進的位置。

例如，如果酒精標籤類別的基本型號假陰性率為 15，而轉接器假陰性率為 15 或更高，您就知道在建立新轉接器時應該專注於新增更多含有酒精標籤的映像。

使用 Rekognition API 作業時，會在呼叫版本作業時傳回 F1 分數量度。DescribeProject

改善模型

轉接器部署是一個反覆的程序，因為您可能需要多次訓練轉接器，才能達到目標的準確度。建立並訓練轉接器之後，您會想要在各種類型的標籤上測試和評估轉接器的效能。

如果您的轉接器在任何區域都缺乏準確度，請新增這些映像的新範例，以提高轉接器對這些標籤的效能。嘗試為轉接器提供額外、多樣的範例，這些範例會反映其存在困境的情況。為您的轉接器提供具代表性、多樣化的映像，讓它能夠處理各種實際範例。

將新映像新增至訓練組後，請重新訓練轉接器，然後重新評估測試集和標籤。重複此程序，直到轉接器達到您想要的效能等級為止。如果您提供更具代表性的映像和註釋，則誤報和誤負評分。將在連續的訓練迭代中逐漸改進。

清單檔案格式

以下各節顯示輸入、輸出和評估檔案的資訊清單檔案格式範例。

輸入資訊清單檔案

資訊清單檔案是以 JSON 線分隔的檔案，每一行都包含一個 JSON，其中包含單一映像的相關資訊。

輸入資訊清單檔案中的每個專案都必須包含具有 Amazon S3 儲存貯體中映像路徑的 `source-ref` 欄位，而對於自訂管制，則包含基本註釋的 `content-moderation-groundtruth` 欄位。一個資料集中的所有映像都預期位於同一個儲存貯體中。訓練和測試資訊清單檔案都是通用的結構。

自訂管制的 CreateProjectVersion 操作會使用「輸入資訊清單」中提供的資訊來訓練轉接器。

下列範例是包含不安全類別之單一映像的資訊清單檔案的一行：

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            }
        ]
   }
}
```

下列範例是單一不安全映像的資訊清單檔案，其中包含多個不安全類別，特別是裸露和粗魯手勢。

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            },
            {
                "Name": "Nudity"
            }
        ]
   }
}
```

下列範例是不包含任何不安全類別的單一映像的資訊清單檔案的一行：

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": []
   }
}
```

如需支援標籤的完整清單，請參閱內容管制。

輸出資訊清單檔案

訓練工作完成後，會傳回輸出資訊清單檔案。輸出資訊清單檔案是 JSON 線上分隔的檔案，其中每一行都包含儲存單一映像資訊的 JSON。Amazon S3 路徑 OutputManifest 可以從DescribeProjectVersion響應中獲得：

- `TrainingDataResult.Output.Assets[0].GroundTruthManifest.S3Object` 用於訓練資料集
- `TestingDataResult.Output.Assets[0].GroundTruthManifest.S3Object` 用於測試資料集

會針對「輸出資訊清單」中的每個項目傳回下列資訊：

鍵值名稱	描述
`source-ref`	引用 s3 中提供的圖像，該圖像在輸入主體中提供
`content-moderation-groundtruth`	輸入清單中提供的地面真相註釋
`detect-moderation-labels`	介面卡預測，僅屬於測試資料集的一部分
`detect-moderation-labels-base-model`	基本模型預測，僅屬於測試資料集的一部分

介面卡和基礎模型預測會以 ConfidenceTrehsold 5.0 的格式傳回，格式類似於「標DetectModeration籤」回應。

下列範例顯示轉接器和基礎模型預測的結構：

```
{
   "ModerationLabels": [
      {
         "Confidence": number,
         "Name": "string",
         "ParentName": "string"
      }
```

```
    ],
    "ModerationModelVersion": "string",
    "ProjectVersion": "string"
}
```

如需傳回標籤的完整清單，請參閱內容管制。

評估結果資訊清單

訓練工作完成後，會傳回評估結果資訊清單檔案。評估結果資訊清單是訓練工作所輸出的 JSON 檔案，其中包含轉接器對測試資料執行情況的相關資訊。

Amazon S3 評估結果的路徑資訊清單可從 DescribeProejctVersion 回應中的現`EvaluationResult.Summary.S3Object`場取得。

下列範例說明評估結果的資訊清單檔案結構：

```
{
    "AggregatedEvaluationResults": {
       "F1Score": number
    },

    "EvaluationDetails": {
        "EvaluationEndTimestamp": "datetime",
        "Labels": [
            "string"
        ],
        "NumberOfTestingImages": number,
        "NumberOfTrainingImages": number,
        "ProjectVersionArn": "string"
    },

    "ContentModeration": {
        "InputConfidenceThresholdEvalResults": {
            "ConfidenceThreshold": float,
            "AggregatedEvaluationResults": {
                "BaseModel": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                },
                "Adapter": {
```

```
                "TruePositive": int,
                "TrueNegative": int,
                "FalsePositive": int,
                "FalseNegative": int
            }
        },
        "LabelEvaluationResults": [
            {
                "Label": "string",
                "BaseModel": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                },
                "Adapter": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                }
            }
        ]
    }
    "AllConfidenceThresholdsEvalResults": [
        {
            "ConfidenceThreshold": float,
            "AggregatedEvaluationResults": {
                "BaseModel": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                },
                "Adapter": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                }
            },
            "LabelEvaluationResults": [
                {
                   "Label": "string",
```

```
                        "BaseModel": {
                            "TruePositive": int,
                            "TrueNegative": int,
                            "FalsePositive": int,
                            "FalseNegative": int
                        },
                        "Adapter": {
                            "TruePositive": int,
                            "TrueNegative": int,
                            "FalsePositive": int,
                            "FalseNegative": int
                        }
                    }
                ]
            }
        ]
    }
}
```

評估資訊清單檔案包含：

- F1Score 定義的彙總結果
- 評估工作的詳細資料 ProjectVersionArn，包括訓練影像的數量、測試影像的數量，以及介面卡訓練所依據的標籤。
- 基礎模型和介面卡效能的彙總 TruePositive FalsePositive、、和 FalseNegative 結果。 TrueNegative
- 基礎模型和介面卡效能的每個標籤 TruePositive、、和 FalseNegative結果，根據輸入可信度閾值計算。 TrueNegative FalsePositive
- 基礎模型和介面卡效能在不同信賴閾值下的彙總和每個標 TruePositive籤 FalsePositive、、和 FalseNegative 結果。 TrueNegative可信度閾值的範圍從 5 到 100，增值幅度為 5。

訓練轉接器的最佳實務

建議您在建立、訓練和使用轉接器時遵守最佳作法：

1. 範例映像資料應擷取客戶想要隱藏的代表性錯誤。如果模型在視覺上相似的映像上重複犯錯誤，請確保帶上許多此類映像進行培訓。

2. 而不是只引入模型在特定管制標籤上錯誤的映像，而是要確保引入映像，表示該模型不會在該管制標籤上發生錯誤。
3. 至少提供 50 個假陰性樣本或 20 個假陽性樣本進行培訓，並提供至少 20 個樣本進行測試。但是，請盡可能提供許多帶註釋的映像，以獲得更好的轉接器性能。
4. 為所有映像重要的所有標籤加上註解：如果您決定需要為映像上的標籤出現加上註解，請務必在所有其他映像上為此標籤出現的專案加上註解。
5. 範例映像資料應在標籤上包含盡可能多的變化，重點放在代表生產設定中分析之映像的執行個體。

設定 AutoUpdate權限

Rekognition 支援自訂介面卡的 AutoUpdate 功能。這意味著在項目上啟用 AutoUpdate 標誌時，自動化再培訓將獲得最大的努力嘗試。這些自動更新需要存取訓練/測試資料集的權限，以及用來訓練客戶介面卡的 AWS KMS 金鑰。您可以按照以下步驟提供這些許可。

Amazon S3 儲存貯體許可

根據預設，所有 Amazon S3 儲存貯體和物件皆為私有。只有資源擁有者 (即建立值區的 AWS 帳號) 可以存取值區及其中包含的任何物件。不過，資源擁有者可藉由編寫儲存貯體政策，選擇將存取許可授予其他資源或使用者。

如果您要建立或修改 Amazon S3 儲存貯體以作為輸入資料集的來源，並在自訂轉接器訓練中使用訓練目的地，則需進一步修改儲存貯體政策。若要讀取或寫入 Amazon S3 儲存貯體，Rekognition 必須具備下列許可。

需要 Rekognition Amazon S3 政策

Rekognition 需要具有下列屬性的許可原則：

- 陳述式 (SID)
- 儲存貯體名稱
- Rekognition 的服務主體名稱。
- Rekognition、儲存貯體及其所有內容所需的資源
- Rekognition 需要採取的必要動作。

下列政策允許 Rekognition 在自動化再訓練期間存取 Amazon S3 儲存貯體。

```
{
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "AllowRekognitionAutoUpdateActions",
            "Principal": {
                "Service": "rekognition.amazonaws.com"
            },
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:HeadObject",
                "s3:HeadBucket"
            ],
            "Resource": [
                "arn:aws:s3:::myBucketName",
                "arn:aws:s3:::myBucketName/*"
            ]
        }
    ]
}
```

您可以按照本指南將上述儲存貯體政策新增至 S3 儲存貯體。

請參閱 此 儲存貯體政策的詳細資訊。

AWS KMS 金鑰權限

Rekognition 可讓您 KmsKeyId 在訓練自訂介面卡時提供選用項目。若有提供，Rekognition 會使用此金鑰來加密複製到服務中的訓練和測試映像，以進行模型訓練。此金鑰也可用來加密寫入輸出 Amazon S3 儲存貯體 (OutputConfig) 的訓練結果和資訊清單檔案。

如果您選擇提供 KMS 金鑰做為自訂轉接器訓練 (亦即`Rekognition:CreateProjectVersion`) 的輸入，則必須進一步修改 KMS 金鑰政策，以允許 Rekognition 服務主體在未來使用此金鑰進行自動重新訓練。Rekognition 必須具有下列許可。

Rekognition 必要金鑰原則 AWS KMS

Amazon Rekognition 需要具有下列屬性的許可政策：

- 陳述式 (SID)
- Amazon Rekognition 的服務主體名稱。

- Amazon Rekognition 需要採取的必要動作。

下列金鑰政策允許 Amazon Rekognition 訓練期間存取 Amazon KMS 金鑰：

```
{
  "Version": "2023-10-06",
  "Statement": [
    {
      "Sid": "KeyPermissions",
      "Effect": "Allow",
      "Principal": {
            "Service": "rekognition.amazonaws.com"
        },
      "Action": [
        "kms:DescribeKey",
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ]
      "Resource": "*"
    }
  ]
}
```

您可以按照本指南將上述 AWS KMS 策略添加到您的 AWS KMS 密鑰中。

請參閱此處有關 AWS KMS 政策的詳細資訊。

AWS Rekognition 的 Health 儀表板名稱

您的 AWS Health 情況儀表板可為來自 Rekognition 的通知提供支援。這些通知提供 Rekognition 模型中可能會影響您應用程式排程變更的感知和補救指引。目前只有 Rekognition 內容管制特徵特定的事件可用。

AWS Health 儀表板是 He AWS alth 服務的一部分。其不需要設定，而且您帳戶中經過驗證的任何使用者皆可檢視。如需更多資訊，請參閱 AWS Health 儀板表入門。

如果您收到的訊息類似以下訊息，應該將其視為警示，以便採取動作。

範例通知：新的模型版本可用於 Rekognition 內容管制。

Rekognition 會將`AWS_MODERATION_MODEL_VERSION_UPDATE_NOTIFICATION`事件發佈至 AWS Health 儀表板，以指出已發行新版本的協調模型。如果您要搭配此 API 使用 DetectModerationLabels

API 和介面卡，此事件非常重要。根據您的使用案例，新模型可能會影響品質，最終會取代先前的模型版本。建議您驗證模型品質，並在收到此警示時瞭解模型更新時間表。

如果您收到模型版本更新通知，則應將其視為要採取動作的警示。如果您不使用轉接器，則應該根據現有使用案例評估更新模型的品質。如果您使用轉接器，您應該使用更新的型號來訓練新的轉接器，並評估其品質。如果您有自動傳輸設定，新的轉接器將會自動進行訓練，然後可以評估其品質。

```
{
   "version": "0",
    "id": "id-number",
    "detail-type": "AWS Health Event",
    "source": "aws.health",
    "account": "123456789012",
    "time": "2023-10-06T06:27:57Z",
    "region": "region",
    "resources": [],
    "detail": {
        "eventArn": "arn:aws:health:us-east-1::event/
AWS_MODERATION_MODEL_UPDATE_NOTIFICATION_event-number",
        "service": "Rekognition",
        "eventTypeCode": "AWS_MODERATION_MODEL_VERSION_UPDATE_NOTIFICATION",
        "eventScopeCode": "ACCOUNT_SPECIFIC",
        "communicationId": "communication-id-number",
        "eventTypeCategory": "scheduledChange",
        "startTime": "Fri, 05 Apr 2023 12:00:00 GMT",
        "lastUpdatedTime": "Fri, 05 Apr 2023 12:00:00 GMT",
        "statusCode": "open",
        "eventRegion": "us-east-1",
        "eventDescription": [
            {
                "language": "en_US",
                "latestDescription": "A new model version is available for Rekognition
 Content Moderation."
            }
        ]
    }
}
```

請參閱使用 Amazon 監控 AWS Health 事件， EventBridge以偵測並回應使用的 AWS Health 事件 EventBridge。

使用 Amazon Augmented AI 檢閱不適當的內容

Amazon Augmented AI (Amazon A2I) 可讓您建立人工檢閱機器學習預測所需的工作流程。

Amazon Rekognition 直接與 Amazon A2I 整合，因此您可以輕鬆對偵測不安全影像的使用案例實作人工檢閱。Amazon A2I 提供對於影像仲裁的人工檢閱工作流程。這可讓您輕鬆檢閱 Amazon Rekognition 的預測。您可以定義使用案例的信賴度閾值，並隨著時間調整它們。使用 Amazon A2I，您可以在自己的組織或 Amazon Mechanical Turk 內使用一組審閱者。您也可以使用 AWS 預先篩選的人力廠商，以確保品質並嚴守安全程序。

以下步驟將引導您完成如何使用亞馬遜 Rekognition 設置亞馬遜 A2I。首先，使用 Amazon A2I 建立流程定義，該流程定義具有觸發人工檢閱的條件。然後，您將流程定義的亞馬遜資源名稱 (ARN) 傳遞給 Amazon Reko `DetectModerationLabel` gnition 操作。在 `DetectModerationLabel` 回應中，您可以查看是否需要人工檢閱。人工審查的結果可在由流程定義設定的 Amazon S3 儲存貯體中取得。

若要檢視如何將 Amazon A2I 與 Amazon Rekognition 搭配使用的 end-to-end 示範，請參閱亞馬遜開發人員指南中的下列其中一個教學課程。 SageMaker

- 示範：在亞馬遜 A2I 主控台中開始使用
- 示範：開始使用亞馬遜 A2I API

  若要開始使用 API，您也可以執行範例 Jupyter 筆記本。請參閱搭配 Amazon A2I Jupyter SageMaker 筆記本使用筆記本執行個體，以便在筆記本執行個體中使用筆記本亞馬遜 Augmented AI (Amazon A2I) 與 Amazon Rekognition 整合 [範例]。 SageMaker

DetectModerationLabels 與亞馬遜 A2I 一起運行

> **Note**
>
> 在同一個 AWS 區域中建立所有的 Amazon A2I 資源和亞馬遜重新認知資源。

1. 完成說明文件中 Amazon Augmented AI 入門中列出的先決條SageMaker 件。

   此外，請記得設定 IAM 許可，如同SageMaker 說明文件中的 Amazon Augmented AI 頁面中的許可和安全性。
2. 遵循SageMaker文件中關於建立人工審核工作流程的指示。

人工檢閱工作流程可管理影像的處理。它包含觸發人工審核的條件、將映像傳送至的工作團隊、工作團隊使用的 UI 範本，以及工作團隊結果傳送至的 Amazon S3 儲存貯體。

在您的CreateFlowDefinition通話中，您需要將設置為「AWS Rekognition///圖像/DetectModerationLabelsV3」。HumanLoopRequestSource之後，你需要決定如何設定會觸發人工檢閱的條件。

使用 Amazon Rekognition，您有兩種選擇ConditionType：ModerationLabelConfidenceCheck和. Sampling

ModerationLabelConfidenceCheck 會在仲裁標籤的可信度位於某範圍內時，建立人類迴圈。最後，Sampling 會傳送隨機百分比的處理文件進行人工檢閱。每個 ConditionType 均使用不同的 ConditionParameters 集合來設定人工檢閱的結果。

ModerationLabelConfidenceCheck 有 ConditionParameters ModerationLableName，其中設定了需要人工檢閱的金鑰。此外，它還具有信心，可以設置使用，和 Equals 發送給人工審查的 LessThan百分比範圍。 GreaterThan SamplingRandomSamplingPercentage其中設置了將發送給人工審查的文檔的百分比。

下列程式碼範例是 CreateFlowDefinition 的部分呼叫。如果在「暗示性」標籤上的評分低於 98%，並在「女性泳衣或內衣褲」標籤上超過 95%，則會傳送影像進行人工檢閱。這意味著，如果影像不被認為是暗示性的，但是有一個穿著內衣褲或泳衣的女人，你可以藉由人工檢閱仔細檢查影像。

```
def create_flow_definition():
'''
Creates a Flow Definition resource

Returns:
struct: FlowDefinitionArn
'''
humanLoopActivationConditions = json.dumps(
    {
        "Conditions": [
            {
              "And": [
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
                    "ConditionParameters": {
```

```
                        "ModerationLabelName": "Suggestive",
                        "ConfidenceLessThan": 98
                    }
                },
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
                    "ConditionParameters": {
                        "ModerationLabelName": "Female Swimwear Or Underwear",
                        "ConfidenceGreaterThan": 95
                    }
                }
            ]
        }
    ]
}
)
```

`CreateFlowDefinition` 會傳回 `FlowDefinitionArn`，您會在下一個步驟中呼叫 `DetectModerationLabels` 時使用。

如需詳細資訊，請CreateFlowDefinition參閱 SageMaker API 參考中的。

3. 呼叫 `DetectModerationLabels` 時設定 `HumanLoopConfig` 參數，如同 偵測不適當的映像 中所示。有關具有 `HumanLoopConfig` set `DetectModerationLabels` 呼叫的示例，請參閱步驟 4。

   a. 在 `HumanLoopConfig` 參數內，將 `FlowDefinitionArn` 設定為您在步驟 2 中所建立流量定義的 ARN。

   b. 設定您的 `HumanLoopName`。這應該是區域中唯一的，且必須是小寫的。

   c. (選擇性) 您可以用`DataAttributes`來設定傳遞給 Amazon Rekognition 的映像是否沒有任何可識別個人身分的資訊。您必須設置此參數才能將信息發送到 Amazon Mechanical Turk。

4. 執行 `DetectModerationLabels`。

   下列範例說明如何使用AWS CLI和AWS SDK for Python (Boto3)來執`DetectModerationLabels`行 `HumanLoopConfig` set。

AWS SDK for Python (Boto3)

下列要求範例使用 SDK for Python (Boto3)。有關詳細信息，請參閱檢測-審查標籤在AWS用於 Python 的軟件開發工具包（博託）API 參考。

```
import boto3

rekognition = boto3.client("rekognition", aws-region)

response = rekognition.detect_moderation_labels( \
        Image={'S3Object': {'Bucket': bucket_name, 'Name': image_name}}, \
        HumanLoopConfig={ \
            'HumanLoopName': 'human_loop_name', \
            'FlowDefinitionArn': , "arn:aws:sagemaker:aws-
region:aws_account_number:flow-definition/flow_def_name" \
            'DataAttributes': {'ContentClassifiers':
 ['FreeOfPersonallyIdentifiableInformation','FreeOfAdultContent']}
         })
```

AWS CLI

下列請求範例使用 AWS CLI。如需詳細資訊，請參閱 AWS CLI 命令參考中的 detect-moderation-labels。

```
$ aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket='bucket_name',Name='image_name'}" \
    --human-loop-config
 HumanLoopName="human_loop_name",FlowDefinitionArn="arn:aws:sagemaker:aws-
region:aws_account_number:flow-
definition/
flow_def_name",DataAttributes='{ContentClassifiers=["FreeOfPersonallyIdentifiableInforma
 "FreeOfAdultContent"]}'
```

```
$ aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket='bucket_name',Name='image_name'}" \
    --human-loop-config \
        '{"HumanLoopName": "human_loop_name", "FlowDefinitionArn":
 "arn:aws:sagemaker:aws-region:aws_account_number:flow-
definition/flow_def_name", "DataAttributes": {"ContentClassifiers":
 ["FreeOfPersonallyIdentifiableInformation", "FreeOfAdultContent"]}}'
```

當您在HumanLoopConfig SageMaker 啟用DetectModerationLabels的情況下執行時，Amazon Rekognition 會呼叫 API 作業。StartHumanLoop此命令會從 DetectModerationLabels 中取得回應，並根據範例中的流量定義條件來檢查它。如果它符合檢閱的條件，則會傳回 HumanLoopArn。這表示您在流程定義中設定的工作小組成員現在可以檢閱映像。呼叫 Amazon Augmented AI 執行時間操作 DescribeHumanLoop 會提供迴圈結果的相關資訊。如需詳細資訊，請參閱 Amazon Augmented AI API 參考文件 DescribeHumanLoop中的。

檢閱影像之後，您可以在流程定義輸出路徑中指定的儲存貯體中查看結果。亞馬遜 A2I 還會在審查完成後通知您亞馬遜 CloudWatch 活動。若要查看要尋找的事件，請參閱文件中的CloudWatch事SageMaker件。

如需詳細資訊，請參閱SageMaker文件中的 Amazon Augmented AI 入門。

偵測文字

Amazon Rekognition 可以偵測映像和影片中的文字。然後，它可以將偵測到的文字轉換為機器可讀取的文字。您可以使用映像中的機器可讀取文字偵測來實作解決方案，例如：

- 視覺搜尋。例如，擷取和顯示包含相同文字的映像。
- 內容見解。例如，針對發生在擷取的影片影格中所辨識到的文字主題提供見解。您的應用程式可以搜尋辨識出的文字以找出相關內容，例如，新聞、運動比數、運動員號碼和字幕。
- 導覽。例如，開發支援語音功能的手機應用程式，協助視障使用者辨識餐廳名稱、商店或街道指標。
- 公共安全與交通協助。例如，從交通監視器映像中偵測車牌號碼。
- 篩選。例如，從映像中篩選出個人身分識別資訊 (PII)。

對於影片中的文字偵測，您可以實作解決方案，例如：

- 搜尋影片中包含特定文字關鍵字的片段，例如新聞節目畫面中的來賓姓名。
- 透過偵測意外文字、褻瀆或垃圾郵件，管制內容以符合組織標準。
- 尋找視訊時間軸上的所有文字浮水印以執行進一步處理，例如用其他語言替換文字以達成內容國際化。
- 尋找文字位置，以便可以相應地對齊其他圖形。

若要偵測 JPEG 或 PNG 格式影像中的文字，請使用DetectText操作。若要以非同步方式偵測視訊中的文字，請使用StartTextDetection和GetTextDetection作業。映像和影片文字偵測作業都支援大多數字型，包括高度風格化的字型。偵測文字之後，Amazon Rekognition 會建立偵測到的字詞和文字行的代表法，顯示兩者間的關聯，並顯示文字在映像或影片影格中的位置。

`DetectText` 和 `GetTextDetection` 操作會偵測文字和線條。單詞是一個或多個不用空格分隔的腳本字符。`DetectText` 最多可以偵測映像中的 100 個單詞。`GetTextDetection` 每幀影片還可以偵測多達 100 個單詞。

單詞是一或多個腳本字符，不以空格分隔。Amazon Rekognition 專為偵測英文、阿拉伯文、俄文、德文、法文、義大利文、葡萄牙文和西班牙文單詞。

文字行是一串等距的字詞。文字行不一定是完整的句子 (句點不表示行的結尾)。例如，Amazon Rekognition 偵測結果為一行文字。當單詞後面沒有對齊或文字之間存在較大的間隙 (相對於字詞的長度) 時，文字行結束。Amazon Rekognition 根據單字間的間隔，可能會偵測以相同方向對齊的多行文字。如果句子跨越多行，此操作會傳回多行文字。

請參考下列映像：

藍色方塊代表 `DetectText` 操作所傳回的偵測到的文字和文字位置的相關資訊。在此範例中，Amazon Rekognition 會將「IT's」、「MONDAY」、「but」、「keep」和「Smiling」偵測為單字。Amazon Rekognition 將「IT's」、「MONDAY」、「but」、「keep」和「Smiling」偵測為文字行。文字必須在水平軸 +/- 90 度方向的範圍內才可被偵測到。

如需範例，請參閱在映像中偵測文字。

主題

- 在映像中偵測文字
- 偵測已儲存影片中的文字

在映像中偵測文字

您提供的輸入映像可以是映像位元組陣列 (Base64 編碼映像位元組)，或者 Amazon S3 物件。在此步驟中，需上傳 jpeg 或 png 映像到 S3 儲存貯體，並指定檔案名稱。

若要偵測映像中的文字 (API)

1. 如果您尚未完成，請先完成事前準備：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS Command Line Interface AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 上傳包含文字的映像到您的 S3 儲存貯體。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列範例來呼叫 DetectText 操作。

   Java

   以下範例程式碼會顯示在映像中偵測到的行和文字。

   將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像名稱。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.DetectTextRequest;
import com.amazonaws.services.rekognition.model.DetectTextResult;
import com.amazonaws.services.rekognition.model.TextDetection;
import java.util.List;



public class DetectText {

   public static void main(String[] args) throws Exception {
```

```
      String photo = "inputtext.jpg";
      String bucket = "bucket";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();



      DetectTextRequest request = new DetectTextRequest()
              .withImage(new Image()
              .withS3Object(new S3Object()
              .withName(photo)
              .withBucket(bucket)));


      try {
         DetectTextResult result = rekognitionClient.detectText(request);
         List<TextDetection> textDetections = result.getTextDetections();

         System.out.println("Detected lines and words for " + photo);
         for (TextDetection text: textDetections) {

                 System.out.println("Detected: " + text.getDetectedText());
                 System.out.println("Confidence: " +
 text.getConfidence().toString());
                 System.out.println("Id : " + text.getId());
                 System.out.println("Parent Id: " + text.getParentId());
                 System.out.println("Type: " + text.getType());
                 System.out.println();
         }
      } catch(AmazonRekognitionException e) {
         e.printStackTrace();
      }
   }
}
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
/**
```

```
*  To run this code example, ensure that you perform the Prerequisites as stated
 in the Amazon Rekognition Guide:
*  https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
*
* Also, ensure that set up your development environment, including your
 credentials.
*
* For information, see this documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/

//snippet-start:[rekognition.java2.detect_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DetectTextRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectTextResponse;
import software.amazon.awssdk.services.rekognition.model.TextDetection;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectTextImage {

 public static void main(String[] args) {

     final String usage = "\n" +
```

```
        "Usage: " +
        "   <sourceImage>\n\n" +
        "Where:\n" +
        "   sourceImage - The path to the image that contains text (for
example, C:\\AWS\\pic1.png). \n\n";

  if (args.length != 1) {
        System.out.println(usage);
        System.exit(1);
    }

    String sourceImage = args[0] ;
    Region region = Region.US_WEST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("default"))
        .build();

    detectTextLabels(rekClient, sourceImage );
    rekClient.close();
}

// snippet-start:[rekognition.java2.detect_text.main]
public static void detectTextLabels(RekognitionClient rekClient, String
sourceImage) {

    try {
        InputStream sourceStream = new FileInputStream(sourceImage);
        SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
        Image souImage = Image.builder()
            .bytes(sourceBytes)
            .build();

        DetectTextRequest textRequest = DetectTextRequest.builder()
            .image(souImage)
            .build();

        DetectTextResponse textResponse = rekClient.detectText(textRequest);
        List<TextDetection> textCollection = textResponse.textDetections();
        System.out.println("Detected lines and words");
        for (TextDetection text: textCollection) {
            System.out.println("Detected: " + text.detectedText());
            System.out.println("Confidence: " + text.confidence().toString());
            System.out.println("Id : " + text.id());
```

```
            System.out.println("Parent Id: " + text.parentId());
            System.out.println("Type: " + text.type());
            System.out.println();
        }

    } catch (RekognitionException | FileNotFoundException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.detect_text.main]
```

AWS CLI

此 AWS CLI 命令會顯示 detect-text CLI 作業的 JSON 輸出。

將 Bucket 與 Name 的數值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像名稱。

使用您開發人員設定檔的名稱取代 profile_name 的值。

```
aws rekognition detect-text  --image "{"S3Object":{"Bucket":"bucket-
name","Name":"image-name"}}" --profile default
```

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。例如，請參閱下列內容：

```
aws rekognition detect-text  --image "{\"S3Object\":{\"Bucket\":\"bucket-name\",
\"Name\":\"image-name\"}}" --profile default
```

Python

以下範例程式碼會顯示在映像中偵測到的行和文字。

將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像名稱。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
```

```
def detect_text(photo, bucket):

    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')

    response = client.detect_text(Image={'S3Object': {'Bucket': bucket, 'Name':
 photo}})

    textDetections = response['TextDetections']
    print('Detected text\n----------')
    for text in textDetections:
        print('Detected text:' + text['DetectedText'])
        print('Confidence: ' + "{:.2f}".format(text['Confidence']) + "%")
        print('Id: {}'.format(text['Id']))
        if 'ParentId' in text:
            print('Parent Id: {}'.format(text['ParentId']))
        print('Type:' + text['Type'])
        print()
    return len(textDetections)

def main():
    bucket = 'bucket-name'
    photo = 'photo-name'
    text_count = detect_text(photo, bucket)
    print("Text detected: " + str(text_count))

if __name__ == "__main__":
    main()
```

.NET

以下範例程式碼會顯示在映像中偵測到的行和文字。

將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像名稱。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
public class DetectText
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectTextRequest detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                }
            }
        };

        try
        {
            DetectTextResponse detectTextResponse =
 rekognitionClient.DetectText(detectTextRequest);
            Console.WriteLine("Detected lines and words for " + photo);
            foreach (TextDetection text in detectTextResponse.TextDetections)
            {
                Console.WriteLine("Detected: " + text.DetectedText);
                Console.WriteLine("Confidence: " + text.Confidence);
                Console.WriteLine("Id : " + text.Id);
                Console.WriteLine("Parent Id: " + text.ParentId);
                Console.WriteLine("Type: " + text.Type);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

Node.JS

以下範例程式碼會顯示在映像中偵測到的行和文字。

將 bucket 與 photo 的數值取代為您在步驟 2 中所使用的 S3 儲存貯體名稱與映像名稱。將 region 的值取代為 .aws 憑證中找到的區域。將建立 Rekognition 工作階段的行中 profile_name 值取代為您開發人員設定檔的名稱。

```
var AWS = require('aws-sdk');

const bucket = 'bucket' // the bucketname without s3://
const photo  = 'photo' // the name of file

const config = new AWS.Config({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
})
AWS.config.update({region:'region'});
const client = new AWS.Rekognition();
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
}
client.detectText(params, function(err, response) {
  if (err) {
    console.log(err, err.stack); // handle error if an error occurred
  } else {
    console.log(`Detected Text for: ${photo}`)
    console.log(response)
    response.TextDetections.forEach(label => {
      console.log(`Detected Text: ${label.DetectedText}`),
      console.log(`Type: ${label.Type}`),
      console.log(`ID: ${label.Id}`),
      console.log(`Parent ID: ${label.ParentId}`),
      console.log(`Confidence: ${label.Confidence}`),
      console.log(`Polygon: `)
      console.log(label.Geometry.Polygon)
    }
```

```
    )
  }
});
```

DetectText 操作請求

在 DetectText 操作中，提供 Base64 編碼位元組陣列的輸入映像或是儲存於 Amazon S3 儲存貯體中的映像。以下範例 JSON 請求顯示從 Amazon S3 儲存貯體載入的映像。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "inputtext.jpg"
        }
    }
}
```

篩選條件

按文字區域、大小和可信度分數進行篩選可為您提供更大的靈活性來控製文字偵測輸出。藉由使用感興趣的區域，您可以輕鬆地將文字偵測限制在與您相關的區域，例如，從機器圖片讀取零件編號時，個人檔案相片右上角或相對於參考點的固定位置 。文字週框方塊大小篩選器可用於避免產生嘈雜或無關緊要的小背景文字。最後，文字可信度篩選器讓您可以消除因朦朧或模糊而導致的不可靠結果。

如需篩選器值的相關資訊，請參閱 DetectTextFilters。

您可以使用下列篩選器：

- MinConfidence設定文字偵測的信賴等級。可信度低於此等級的文字會從結果中排除。值應該介於 0 和 100 之間。
- MinBoundingBoxWidth— 設定單字邊界方框的最小寬度。週邊方塊小於此值的文字會從結果中排除。此值相對於映像影格寬度。
- MinBoundingBoxHeight— 設定單字邊界方框的最小高度。週邊方塊高度小於此值的文字會從結果中排除。此值相對於映像影格高度。
- RegionsOfInterest— 將偵測限制在影像框的特定區域。這些值相對於影格尺寸。對於僅部分區域內的文字，回應是未定義的。

DetectText 作業回應

該DetectText操作分析圖像並返回一個數組 TextDetections，其中每個元素（ TextDetection ）代表在圖像中檢測到的行或單詞。針對每個元素，DetectText 會傳回以下資訊：

- 偵測到的文字 (DetectedText)
- 單字與文字行之間的關係 (Id 和 ParentId)
- 文字在映像中的位置 (Geometry)
- Amazon Rekognition 對於文字與週框方塊 (Confidence) 偵測精確度的可信度
- 偵測到的文字類型 (Type)

偵測到的文字

每個 TextDetection 元素包含辨識到的文字 (單字或行) DetectedText 欄位。單字是一或多個指令碼字元，不以空格分隔。DetectText 最多可在一個映像中偵測 100 個單字。傳回的文字可能包含字元，讓單字無法辨識。例如，C@t 而不是 Cat。若要判斷 TextDetection 元素代表的是一行文字或單字，請使用 Type 欄位。

每個 TextDetection 元素都包含百分比值，代表 Amazon Rekognition 對於文字與文字周圍的週框方塊之偵測精確度的可信度。

單字和行的關係

每個 TextDetection 元素都有一個辨識碼欄位，Id。Id 會顯示單字在行中的位置。如果元素是一個單字，父系識別碼欄位 ParentId 將找出偵測到該單字的行。行的 ParentId 為 null。例如，範例映像中的「but keep」文字行具有下列 Id 和 ParentId 值：

文字	ID	父系 ID
但保留	3	
但	8	3
保留	9	3

文字在映像中的位置

若要判定辨識到的文字在映像中的位置，請使用 DetectText 傳回的週框方塊 (幾何圖形) 資訊。Geometry 物件包含用於偵測到的行和單字之兩種類型的週框方塊資訊：

- 物件中軸對齊的粗糙矩形外框 BoundingBox
- 在點陣列中由多個 X 與 Y 座標組成的精密多邊形

週框方塊與多邊形座標將顯示文字在映像中所在的位置。座標值為整體映像大小的比例。如需詳細資訊，請參閱BoundingBox。

下列來自 DetectText 操作的 JSON 回應將顯示在下列映像中偵測到的單字與行。


```
{
 'TextDetections': [{'Confidence': 99.35693359375,
                     'DetectedText': "IT'S",
                     'Geometry': {'BoundingBox': {'Height': 0.09988046437501907,
                                                  'Left': 0.6684935688972473,
                                                  'Top': 0.18226495385169983,
                                                  'Width': 0.1461552083492279},
                                  'Polygon': [{'X': 0.6684935688972473,
```

```
                                    'Y': 0.1838926374912262},
                                   {'X': 0.8141663074493408,
                                    'Y': 0.18226495385169983},
                                   {'X': 0.8146487474441528,
                                    'Y': 0.28051772713661194},
                                   {'X': 0.6689760088920593,
                                    'Y': 0.2821454107761383}]},
                  'Id': 0,
                  'Type': 'LINE'},
                 {'Confidence': 99.6207275390625,
                  'DetectedText': 'MONDAY',
                  'Geometry': {'BoundingBox': {'Height': 0.11442459374666214,
                                               'Left': 0.5566731691360474,
                                               'Top': 0.3525116443634033,
                                               'Width': 0.39574965834617615},
                               'Polygon': [{'X': 0.5566731691360474,
                                            'Y': 0.353712260723114},
                                           {'X': 0.9522717595100403,
                                            'Y': 0.3525116443634033},
                                           {'X': 0.9524227976799011,
                                            'Y': 0.4657355844974518},
                                           {'X': 0.5568241477012634,
                                            'Y': 0.46693623065948486}]},
                  'Id': 1,
                  'Type': 'LINE'},
                 {'Confidence': 99.6160888671875,
                  'DetectedText': 'but keep',
                  'Geometry': {'BoundingBox': {'Height': 0.08314694464206696,
                                               'Left': 0.6398131847381592,
                                               'Top': 0.5267938375473022,
                                               'Width': 0.2021435648202896},
                               'Polygon': [{'X': 0.640289306640625,
                                            'Y': 0.5267938375473022},
                                           {'X': 0.8419567942619324,
                                            'Y': 0.5295097827911377},
                                           {'X': 0.8414806723594666,
                                            'Y': 0.609940767288208},
                                           {'X': 0.6398131847381592,
                                            'Y': 0.6072247624397278}]},
                  'Id': 2,
                  'Type': 'LINE'},
                 {'Confidence': 88.95134735107422,
                  'DetectedText': 'Smiling',
                  'Geometry': {'BoundingBox': {'Height': 0.4326171875,
```

```
                                        'Left': 0.46289217472076416,
                                        'Top': 0.5634765625,
                                        'Width': 0.5371078252792358},
                             'Polygon': [{'X': 0.46289217472076416,
                                          'Y': 0.5634765625},
                                         {'X': 1.0, 'Y': 0.5634765625},
                                         {'X': 1.0, 'Y': 0.99609375},
                                         {'X': 0.46289217472076416,
                                          'Y': 0.99609375}]},
                  'Id': 3,
                  'Type': 'LINE'},
                 {'Confidence': 99.35693359375,
                  'DetectedText': "IT'S",
                  'Geometry': {'BoundingBox': {'Height': 0.09988046437501907,
                                               'Left': 0.6684935688972473,
                                               'Top': 0.18226495385169983,
                                               'Width': 0.1461552083492279},
                               'Polygon': [{'X': 0.6684935688972473,
                                            'Y': 0.1838926374912262},
                                           {'X': 0.8141663074493408,
                                            'Y': 0.18226495385169983},
                                           {'X': 0.8146487474441528,
                                            'Y': 0.28051772713661194},
                                           {'X': 0.6689760088920593,
                                            'Y': 0.2821454107761383}]},
                  'Id': 4,
                  'ParentId': 0,
                  'Type': 'WORD'},
                 {'Confidence': 99.6207275390625,
                  'DetectedText': 'MONDAY',
                  'Geometry': {'BoundingBox': {'Height': 0.11442466825246811,
                                               'Left': 0.5566731691360474,
                                               'Top': 0.35251158475875854,
                                               'Width': 0.39574965834617615},
                               'Polygon': [{'X': 0.5566731691360474,
                                            'Y': 0.3537122905254364},
                                           {'X': 0.9522718787193298,
                                            'Y': 0.35251158475875854},
                                           {'X': 0.9524227976799011,
                                            'Y': 0.4657355546951294},
                                           {'X': 0.5568241477012634,
                                            'Y': 0.46693626046180725}]},
                  'Id': 5,
                  'ParentId': 1,
```

```
                     'Type': 'WORD'},
                    {'Confidence': 99.96778869628906,
                     'DetectedText': 'but',
                     'Geometry': {'BoundingBox': {'Height': 0.0625,
                                                  'Left': 0.6402802467346191,
                                                  'Top': 0.5283203125,
                                                  'Width': 0.08027780801057816},
                                  'Polygon': [{'X': 0.6402802467346191,
                                               'Y': 0.5283203125},
                                              {'X': 0.7205580472946167,
                                               'Y': 0.5283203125},
                                              {'X': 0.7205580472946167,
                                               'Y': 0.5908203125},
                                              {'X': 0.6402802467346191,
                                               'Y': 0.5908203125}]},
                     'Id': 6,
                     'ParentId': 2,
                     'Type': 'WORD'},
                    {'Confidence': 99.26438903808594,
                     'DetectedText': 'keep',
                     'Geometry': {'BoundingBox': {'Height': 0.0818721204996109,
                                                  'Left': 0.7344760298728943,
                                                  'Top': 0.5280686020851135,
                                                  'Width': 0.10748066753149033},
                                  'Polygon': [{'X': 0.7349520921707153,
                                               'Y': 0.5280686020851135},
                                              {'X': 0.8419566750526428,
                                               'Y': 0.5295097827911377},
                                              {'X': 0.8414806127548218,
                                               'Y': 0.6099407076835632},
                                              {'X': 0.7344760298728943,
                                               'Y': 0.6084995269775391}]},
                     'Id': 7,
                     'ParentId': 2,
                     'Type': 'WORD'},
                    {'Confidence': 88.95134735107422,
                     'DetectedText': 'Smiling',
                     'Geometry': {'BoundingBox': {'Height': 0.4326171875,
                                                  'Left': 0.46289217472076416,
                                                  'Top': 0.5634765625,
                                                  'Width': 0.5371078252792358},
                                  'Polygon': [{'X': 0.46289217472076416,
                                               'Y': 0.5634765625},
                                              {'X': 1.0, 'Y': 0.5634765625},
```

```
                                                {'X': 1.0, 'Y': 0.99609375},
                                                {'X': 0.46289217472076416,
                                                 'Y': 0.99609375}]},
                    'Id': 8,
                    'ParentId': 3,
                    'Type': 'WORD'}],
 'TextModelVersion': '3.0'}
```

偵測已儲存影片中的文字

Amazon Rekognition Video 已儲存影片中的文字偵測是一項非同步操作。要開始檢測文本，請致電StartTextDetection。Amazon Rekognition Video 會將影片分析的完成狀态发布至 Amazon SNS 主題。如果視頻分析成功，請致電GetTextDetection以獲取分析結果。如需開始影片分析並取得結果的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

此程序會在 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中展開程式碼。此程序使用 Amazon SQS 佇列來獲取影片分析要求的完成狀態。

偵測存放於 Amazon S3 儲存貯體 (SDK) 之影片中的文字

1. 請執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中的步驟。
2. 將下列程式碼加入至步驟 1 中的類別 `VideoDetect`。

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)


private static void StartTextDetection(String bucket, String video) throws
 Exception{

    NotificationChannel channel= new NotificationChannel()
            .withSNSTopicArn(snsTopicArn)
            .withRoleArn(roleArn);

    StartTextDetectionRequest req = new StartTextDetectionRequest()
            .withVideo(new Video()
                    .withS3Object(new S3Object()
                        .withBucket(bucket)
```

```
                        .withName(video)))
            .withNotificationChannel(channel);


    StartTextDetectionResult startTextDetectionResult =
 rek.startTextDetection(req);
    startJobId=startTextDetectionResult.getJobId();

}

private static void GetTextDetectionResults() throws Exception{

    int maxResults=10;
    String paginationToken=null;
    GetTextDetectionResult textDetectionResult=null;

    do{
        if (textDetectionResult !=null){
            paginationToken = textDetectionResult.getNextToken();

        }


        textDetectionResult = rek.getTextDetection(new
 GetTextDetectionRequest()
             .withJobId(startJobId)
             .withNextToken(paginationToken)
             .withMaxResults(maxResults));

        VideoMetadata videoMetaData=textDetectionResult.getVideoMetadata();

        System.out.println("Format: " + videoMetaData.getFormat());
        System.out.println("Codec: " + videoMetaData.getCodec());
        System.out.println("Duration: " + videoMetaData.getDurationMillis());
        System.out.println("FrameRate: " + videoMetaData.getFrameRate());


        //Show text, confidence values
        List<TextDetectionResult> textDetections =
 textDetectionResult.getTextDetections();


        for (TextDetectionResult text: textDetections) {
            long seconds=text.getTimestamp()/1000;
```

```
            System.out.println("Sec: " + Long.toString(seconds) + " ");
            TextDetection detectedText=text.getTextDetection();

            System.out.println("Text Detected: " +
 detectedText.getDetectedText());
                System.out.println("Confidence: " +
 detectedText.getConfidence().toString());
                System.out.println("Id : " + detectedText.getId());
                System.out.println("Parent Id: " + detectedText.getParentId());
                System.out.println("Bounding Box" +
 detectedText.getGeometry().getBoundingBox().toString());
                System.out.println("Type: " + detectedText.getType());
                System.out.println();
        }
    } while (textDetectionResult !=null && textDetectionResult.getNextToken() !=
 null);


}
```

在函數 `main` 中，將下行:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

取代為：

```
        StartTextDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetTextDetectionResults();
```

Java V2

此代碼取自 AWS 文檔 SDK 示例 GitHub 存儲庫。請參閱此處的完整範例。

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
```

```
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectTextVideo {

 private static String startJobId ="";
 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
         "   bucket - The name of the bucket in which the video is located (for
 example, (for example, myBucket). \n\n"+
         "   video - The name of video (for example, people.mp4). \n\n" +
         "   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic. \n\n" +
         "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;

     if (args.length != 4) {
```

```
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    startTextLabels(rekClient, channel, bucket, video);
    GetTextResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_text.main]
public static void startTextLabels(RekognitionClient rekClient,
                                   NotificationChannel channel,
                                   String bucket,
                                   String video) {
    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();

        Video vidOb = Video.builder()
            .s3Object(s3Obj)
            .build();

        StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
            .jobTag("DetectingLabels")
```

```
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
        startJobId = labelDetectionResponse.jobId();

    } catch (RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}

public static void GetTextResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetTextDetectionResponse textDetectionResponse=null;
        boolean finished = false;
        String status;
        int yy=0 ;

        do{
            if (textDetectionResponse !=null)
                paginationToken = textDetectionResponse.nextToken();

            GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                .jobId(startJobId)
                .nextToken(paginationToken)
                .maxResults(10)
                .build();

            // Wait until the job succeeds.
            while (!finished) {
                textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                status = textDetectionResponse.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
```

```
                    Thread.sleep(1000);
                }
                yy++;
            }

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null.
            VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<TextDetectionResult> labels=
 textDetectionResponse.textDetections();
            for (TextDetectionResult detectedText: labels) {
                System.out.println("Confidence: " +
 detectedText.textDetection().confidence().toString());
                System.out.println("Id : " +
 detectedText.textDetection().id());
                System.out.println("Parent Id: " +
 detectedText.textDetection().parentId());
                System.out.println("Type: " +
 detectedText.textDetection().type());
                System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                System.out.println();
            }

        } while (textDetectionResponse !=null &&
 textDetectionResponse.nextToken() != null);

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_text.main]
}
```

Python

```
#Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def StartTextDetection(self):
        response=self.rek.start_text_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetTextDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_text_detection(JobId=self.startJobId,
                                        MaxResults=maxResults,
                                        NextToken=paginationToken)

            print('Codec: ' + response['VideoMetadata']['Codec'])

            print('Duration: ' + str(response['VideoMetadata']
['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for textDetection in response['TextDetections']:
                text=textDetection['TextDetection']

                print("Timestamp: " + str(textDetection['Timestamp']))
                print("   Text Detected: " + text['DetectedText'])
                print("   Confidence: " +  str(text['Confidence']))
                print ("      Bounding box")
                print ("        Top: " + str(text['Geometry']['BoundingBox']
['Top']))
```

```
                print ("        Left: " + str(text['Geometry']['BoundingBox']
['Left']))
                print ("        Width: " +  str(text['Geometry']['BoundingBox']
['Width']))
                print ("        Height: " +  str(text['Geometry']['BoundingBox']
['Height']))
                print ("   Type: " + str(text['Type']) )
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

在函數 main 中，將下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

取代為：

```
    analyzer.StartTextDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetTextDetectionResults()
```

CLI

運行以下 AWS CLI 命令以開始檢測視頻中的文本。

```
 aws rekognition start-text-detection --video "{"S3Object":{"Bucket":"bucket-
name","Name":"video-name"}}"\
 --notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
 --region region-name --profile profile-name
```

更新下列的值：

- 將 bucket-name 與 video-name 變更為您在步驟 2 中指定的 Amazon S3 儲存貯體與文檔名稱。
- 將 region-name 變更為您正在使用的 AWS 區域。

- 使用您開發人員設定檔的名稱取代 `profile-name` 的值。
- 將 `topic-ARN` 變更為您在 設定 Amazon Rekognition Video 步驟 3 建立的 Amazon SNS 主題 ARN。
- 將 `role-ARN` 變更為您在步驟 7 建立的 設定 Amazon Rekognition Video IAM 服務角色的 ARN。

如果您在 Windows 裝置上存取 CLI，請使用雙引號而非單引號，並以反斜線 (即\) 替代內部雙引號，以解決您可能遇到的任何剖析器錯誤。如需範例，請參閱下列內容：

```
aws rekognition start-text-detection --video \
 "{\"S3Object\":{\"Bucket\":\"bucket-name\",\"Name\":\"video-name\"}}" \
 --notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
 --region region-name --profile profile-name
```

執行正在進行的程式碼範例之後，複製傳回的程式碼 `jobID`，並將其提供給下列 `GetTextDetection` 命令，以 `job-id-number` 取代您之前收到的 `jobID` 結果：

```
aws rekognition get-text-detection --job-id job-id-number --profile profile-name
```

> **Note**
>
> 如果您已執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 以外的影片範例，要取代的程式碼可能會不同。

3. 執行程式碼。在影片中偵測到的文字會顯示在清單中。

篩選條件

篩選器是可選的要求參數，當您呼叫 `StartTextDetection` 時，會使用此篩選器。按文字區域、大小和可信度分數進行篩選可為您提供更大的靈活性來控製文字偵測輸出。透過感興趣的區域，您可以輕鬆地將文字偵測限制在與您相關的區域，例如，圖形的底部第三區域或足球遊戲中讀取記分牌的左上角。文字週框方塊大小篩選器可用於避免產生嘈雜或無關緊要的小背景文字。最後，文字可信度篩選器讓您可以消除因朦朧或模糊而導致的不可靠結果。

如需篩選器值的相關資訊，請參閱 DetectTextFilters。

您可以使用下列篩選器：

- MinConfidence設定文字偵測的信賴等級。可信度低於此等級的文字會從結果中排除。值應該介於 0 和 100 之間。
- MinBoundingBoxWidth— 設定單字邊界方框的最小寬度。週邊方塊小於此值的文字會從結果中排除。此值相對於影片影格寬度。
- MinBoundingBoxHeight— 設定單字邊界方框的最小高度。週邊方塊高度小於此值的文字會從結果中排除。此值相對於影片影格高度。
- RegionsOfInterest— 將偵測限制在框架的特定區域。這些值是對於影格尺寸。對於僅部分區域內的物件，回應是未定義的。

GetTextDetection 回應

GetTextDetection 會傳回陣列 (TextDetectionResults)，其中包含影片中偵測到之文字的相關資訊。每當在影片中偵測到文字或文字行時，就會有一個陣列元素 TextDetection。陣列元素會從影片開頭起依時間 (以毫秒為單位) 排序。

以下是 GetTextDetection 的部分 JSON 回應。在回應中，請注意下列事項：

- 文字資訊 — TextDetectionResult 陣列元素包含偵測到的文字 (TextDetection) 以及在 video (Timestamp) 中偵測到文字的時間的相關資訊。
- 分頁資訊：該範例顯示一頁文字偵測資訊。您可以在 GetTextDetection 的 MaxResults 輸入參數中指定要傳回的文字元素數目。如果結果數目超過 MaxResults，或結果數目超過預設最大值，GetTextDetection 會傳回用來取得下一頁結果的字符 (NextToken)。如需詳細資訊，請參閱 取得 Amazon Rekognition Video 分析結果。
- 影片資訊：回應包含 VideoMetadata 所傳回之每頁資訊中影片格式 (GetTextDetection) 的相關資訊。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 174441,
        "Format": "QuickTime / MOV",
```

```
        "FrameRate": 29.970029830932617,
        "FrameHeight": 480,
        "FrameWidth": 854
    },
    "TextDetections": [
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle Twinkle Little Star",
                "Type": "LINE",
                "Id": 0,
                "Confidence": 99.91780090332031,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.8337579369544983,
                        "Height": 0.08365312218666077,
                        "Left": 0.08313830941915512,
                        "Top": 0.4663468301296234
                    },
                    "Polygon": [
                        {
                            "X": 0.08313830941915512,
                            "Y": 0.4663468301296234
                        },
                        {
                            "X": 0.9168962240219116,
                            "Y": 0.4674469828605652
                        },
                        {
                            "X": 0.916861355304718,
                            "Y": 0.5511001348495483
                        },
                        {
                            "X": 0.08310343325138092,
                            "Y": 0.5499999523162842
                        }
                    ]
                }
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle",
```

```
                "Type": "WORD",
                "Id": 1,
                "ParentId": 0,
                "Confidence": 99.98338317871094,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.2423887550830841,
                        "Height": 0.0833333358168602,
                        "Left": 0.08313817530870438,
                        "Top": 0.46666666865348816
                    },
                    "Polygon": [
                        {
                            "X": 0.08313817530870438,
                            "Y": 0.46666666865348816
                        },
                        {
                            "X": 0.3255269229412079,
                            "Y": 0.46666666865348816
                        },
                        {
                            "X": 0.3255269229412079,
                            "Y": 0.550000011920929
                        },
                        {
                            "X": 0.08313817530870438,
                            "Y": 0.550000011920929
                        }
                    ]
                }
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle",
                "Type": "WORD",
                "Id": 2,
                "ParentId": 0,
                "Confidence": 99.982666015625,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.2423887550830841,
                        "Height": 0.08124999701976776,
```

```
                    "Left": 0.3454332649707794,
                    "Top": 0.46875
                },
                "Polygon": [
                    {
                        "X": 0.3454332649707794,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.5878220200538635,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.5878220200538635,
                        "Y": 0.550000011920929
                    },
                    {
                        "X": 0.3454332649707794,
                        "Y": 0.550000011920929
                    }
                ]
            }
        }
    },
    {
        "Timestamp": 967,
        "TextDetection": {
            "DetectedText": "Little",
            "Type": "WORD",
            "Id": 3,
            "ParentId": 0,
            "Confidence": 99.8787612915039,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.16627635061740875,
                    "Height": 0.08124999701976776,
                    "Left": 0.6053864359855652,
                    "Top": 0.46875
                },
                "Polygon": [
                    {
                        "X": 0.6053864359855652,
                        "Y": 0.46875
                    },
```

```
                    {
                        "X": 0.7716627717018127,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.7716627717018127,
                        "Y": 0.550000011920929
                    },
                    {
                        "X": 0.6053864359855652,
                        "Y": 0.550000011920929
                    }
                ]
            }
        }
    },
    {
        "Timestamp": 967,
        "TextDetection": {
            "DetectedText": "Star",
            "Type": "WORD",
            "Id": 4,
            "ParentId": 0,
            "Confidence": 99.82640075683594,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.12997658550739288,
                    "Height": 0.08124999701976776,
                    "Left": 0.7868852615356445,
                    "Top": 0.46875
                },
                "Polygon": [
                    {
                        "X": 0.7868852615356445,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.9168618321418762,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.9168618321418762,
                        "Y": 0.550000011920929
                    },
```

```
                        {
                            "X": 0.7868852615356445,
                            "Y": 0.550000011920929
                        }
                    ]
                }
            }
        }
    ],
    "NextToken": "NiHpGbZFnkM/S8kLcukMni15wb05iKtquu/Mwc+Qg1LVlMjjKNOD0Z0GusSPg7TONLe
+OZ3P",
    "TextModelVersion": "3.0"
}
```

偵測已存放影片中的影片區段

Amazon Rekognition Video 提供的 API 可識別有用的影片片段，例如黑框和片尾名單。

觀眾現在觀看的內容比以往都多。特別是，Over-The-Top (OTT) 和隨需影片 (VOD) 平台隨時隨地在任何螢幕上都能提供豐富的內容選擇。隨著內容量不斷增加，媒體公司在準備和管理內容方面面臨挑戰。這對於提供高品質的觀賞體驗和更好的營利內容而言，至關重要。如今，公司使用由訓練有素的人力所組成的大型團隊來執行下列任務。

- 在一段內容中找到片頭片尾名單的開始位置。
- 選擇正確的點來插入廣告，例如靜音黑色影格序列
- 將影片分割成較小的剪輯片段，以便進行索引。

這些手動程序既昂貴又緩慢，而且規模無法跟上每天從封存製作、授權和擷取的內容量。

您可以使用 Amazon Rekognition Video，採用機器學習 (ML) 支援的全受管、專用影片區段偵測 API 來自動執行操作媒體分析任務。透過使用 Amazon Rekognition Video 區段 API，您可以輕鬆分析大量影片，並偵測黑色影格或鏡頭轉換等標記。您可以取得每次偵測的 SMPTE (電影電視工程師協會) 時間碼、時間戳記和影格數量。不需要機器學習經驗。

Amazon Rekognition Video 影片會分析 Amazon Simple Storage Service (Amazon S3) 儲存貯體中存放的影片。Amazon Rekognition Video 傳回的 SMPTE 是影格準確的時間碼：提供偵測到的影片區段精確影格數，並處理各種影片影格播放速率格式。您可以使用來自 Amazon Rekognition Video 的影格準確中繼資料，將特定任務完全自動化，或大幅降低受訓人工操作員的檢閱工作負載，使其可著重在更有創造性的工作。這可讓您執行諸如準備內容、插入廣告以及在雲端中，大規模地將「狂歡標記」新增至內容等任務。

如需定價的相關資訊，請參閱Amazon Rekognition 定價。

Amazon Rekognition Video 區段偵測支援兩種類型的分段任務：技術提示 偵測和 鏡頭偵測。

主題

- 技術提示
- 鏡頭偵測
- 關於 Amazon Rekognition Video 區段偵測 API
- 使用 Amazon Rekognition 區段 API

- 範例：偵測已存放影片中的區段

技術提示

技術提示可識別影片中的黑色框架、色彩導表、片頭字幕、片尾名單、工作室標誌和主要節目內容。

黑色畫面

影片通常包含一小段無音訊的黑色空白畫面，用作插入廣告或劃定場景或開場演職員表等節目段結尾的提示。借助 Amazon Rekognition Video，您可以偵測此類黑色畫面序列，以自動執行廣告插入、為 VOD 打包內容並劃分各種節目段或場景。帶有音訊的黑色畫面 (例如淡出或旁白) 視為內容，且不會傳回。

Credits (點數)

Amazon Rekognition Video 可幫助您自動識別電影或電視節目片尾名單開始和結束的確切畫面。有了這些資訊，您可以在隨選影片 (VOD) 應用程式中產生「狂歡標記」或互動式檢視器提示，例如「下一集」或「跳過簡介」。您還可以偵測影片中的程序內容的第一幀和最後一幀。Amazon Rekognition Video 經過訓練，可處理各種開始和結束信用風格，從簡單的滾動積分到更具挑戰性的信用以及內容。

彩條信號

Amazon Rekognition Video 讓您可以偵測顯示 SMPTE 彩條信號的影片部分，SMPTE 彩條信號是以特定模式顯示的一組顏色，用於確保顏色在廣播監控器、節目和攝影機上正確校正。如需 SMPTE 彩條信號的詳細資訊，請參閱 SMPTE 彩條信號。當彩條信號作為預設信號而不是內容連續顯示時，此中繼資料可用於透過從內容中移除彩條信號來為隨選影片應用程式準備內容，或用於偵測錄製影片中廣播信號遺失等問題。

场记板

场记板是影片的部分，通常位於開頭附近，其中包含有關單集、工作室、視訊格式、音頻頻道等的文字中繼資料。Amazon Rekognition Video 可以識別场记板的開始和結尾，讓您在準備最終檢視的內容時輕鬆使用文字中繼資料或移除场记板。

工作室徽標

工作室徽標是顯示參與製作演出的製作工作室的徽標或標誌的序列。Amazon Rekognition Video 可以偵測這些序列，讓使用者可以檢閱其以識別工作室。

內容

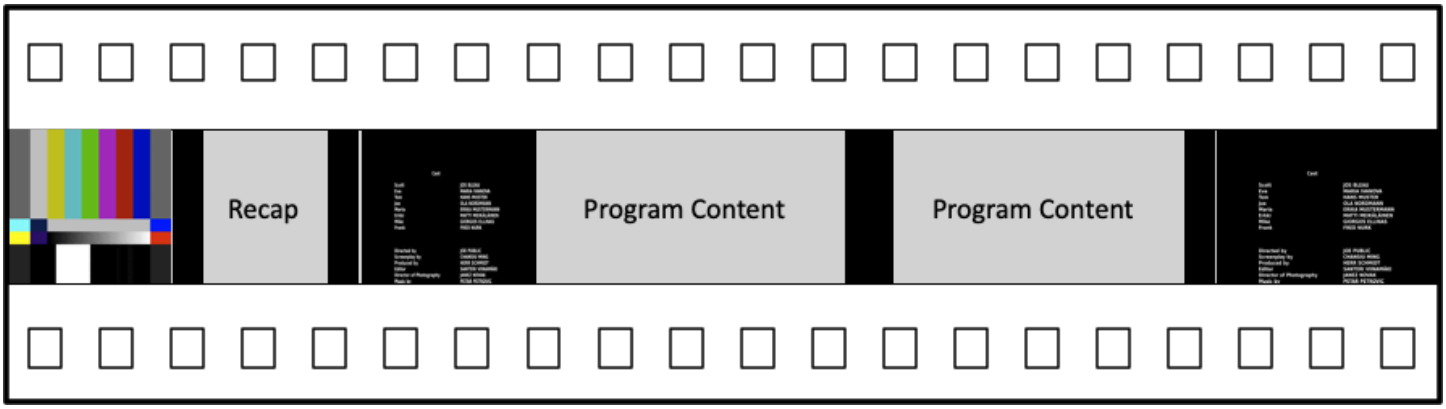
內容是電視節目或電影中包含該節目或相關元素的部分。黑色畫面、片頭片尾名單、彩條信號、场记板和工作室徽標不被視為內容。Amazon Rekognition Video 可以偵測影片中每個內容區段的開始和結束，因此您可以找到程式執行時間或特定區段。

內容區段包括但不限於下列專案：

- 兩個廣告插播之間的程式場景
- 在影片開始時快速回顧上一集
- 獎金信用後內容
- 「無文字」內容，例如一組原本包含重疊文字的所有程式場景，但已移除文字以支援翻譯成其他語言的所有程式場景。

Amazon Rekognition Video 完成所有內容區段偵測之後，您可以套用網域知識或傳送給人工審查，以進一步分類每個區段。舉例來說，如果您使用的影片一律以回顧開頭，您可以將第一個內容區段歸類為回顧。

下圖說明節目或電影時間軸上的技術提示區段。請注意色彩導表和片頭名單資訊、內容區段 (例如回顧和主要程式)、整個影片中的黑框，以及片尾名單資訊。



鏡頭偵測

鏡頭是由一台攝影機連續拍攝的一系列相互關聯的連續映像，代表在時間和空間上的連續動作。借助 Amazon Rekognition Video，您可以偵測每個鏡頭的開始、結束和持續時間，並可以計算內容中的所有鏡頭。您可以使用任務的鏡頭中繼資料，如下所視。

- 使用選取的鏡頭製作宣傳影片。

- 在不會影響觀眾體驗的地方插入廣告，例如，有人說話時的鏡頭中間。
- 產生一組預覽縮圖，避免鏡頭之間的轉場內容。

鏡頭偵測會標記在對其他相機有硬切換的確切畫面上。如果有從某個攝影機到另一個攝影機的柔性轉場，則 Amazon Rekognition Video 會逸出轉場。如此可確保鏡頭開始與結束時間不包含沒有實際內容的區段。

下圖說明電影底片的鏡頭偵測區段。請注意，從某個相機角度或位置到下一個相機角度的鏡頭來識別每個拍攝畫面。

關於 Amazon Rekognition Video 區段偵測 API

若要對已儲存的影片進行區段，您可以使用非同步StartSegmentDetection和GetSegmentDetectionAPI 作業來啟動區段工作並擷取結果。區段偵測可接受存放在 Amazon S3 儲存貯體中的影片，並傳回 JSON 輸出。您可以透過設定 `StartSegmentdetection` API 要求，選擇僅偵測技術提示、僅偵測鏡頭轉換，或兩者兼有。您也可以透過設定最低預測可信度的閾值，來篩選偵測到的區段。如需詳細資訊，請參閱 使用 Amazon Rekognition 區段 API 。如需範例程式碼，請參閱 範例：偵測已存放影片中的區段。

使用 Amazon Rekognition 區段 API

Amazon Rekognition Video 中的區段偵測是一種 Amazon Rekognition Video 非同步作業。Amazon Rekognition 區段 API 是一種複合 API，您可以從單一 API 呼叫中選擇分析類型 (技術提示或鏡頭偵測)。如需呼叫非同步操作的詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

主題

- 啟動區段分析

- 取得區段分析結果

啟動區段分析

開始偵測已儲存視訊通話中的區段StartSegmentDetection。輸入參數與加入區段類型選擇和結果篩選的其他 Amazon Rekognition Video 操作相同。如需詳細資訊，請參閱 開始影片分析。

以下是 StartSegmentDetection 傳遞的 JSON 範例。此要求指定同時偵測技術提示和鏡頭偵測區段。針對技術提示區段 (90%) 和鏡頭偵測區段 (80%)，要求使用不同的篩選條件來達到最低偵測可信度。

```
{
  "Video": {
    "S3Object": {
      "Bucket": "test_files",
      "Name": "test_file.mp4"
    }
    "SegmentTypes":["TECHNICAL_CUES", "SHOT"]
    "Filters": {
      "TechnicalCueFilter": {
         "MinSegmentConfidence": 90,
         "BlackFrame" : {
            "MaxPixelThreshold": 0.1,
            "MinCoveragePercentage": 95
         }
      },
      "ShotFilter" : {
          "MinSegmentConfidence": 60
      }
  }
}
```

選擇區段類型

使用 SegmentTypes 陣列輸入參數來偵測輸入影片中的技術提示及/或鏡頭偵測區段。

- TECHNICAL_CUE — 可針對影片中偵測到的片頭、片尾和期間技術提示 (黑條信號、彩色信號、片頭名單、片尾名單、工作室標誌及原節目內容)，識別影格準確的時間戳記。例如，您可以使用技術提示來尋找片尾名單的開頭。如需詳細資訊，請參閱 技術提示。
- SHOT：可識別鏡頭的開始、結束和持續時間。例如，您可以使用鏡頭偵測來識別影片最後編輯的候選鏡頭。如需詳細資訊，請參閱 鏡頭偵測。

篩選分析結果

您可以使用 Filters (StartSegmentDetectionFilters) input 參數來指定回應中傳回的最小偵測信賴度。在中Filters，使用 ShotFilter (StartShotDetectionFilter) 過濾偵測到的鏡頭。使用 TechnicalCueFilter (StartTechnicalCueDetectionFilter) 篩選技術提示。

如需範例程式碼，請參閱 範例：偵測已存放影片中的區段。

取得區段分析結果

Amazon Rekognition Video 向 Amazon Simple Notification Service 主題發佈影片的完成狀態。如果視頻分析成功，請致電GetSegmentDetection以獲取視頻分析的結果。

以下是範例 GetSegmentDetection 要求。JobId 是呼叫 StartSegmentDetection 傳回的任務識別符。如需其他輸入參數的詳細資訊，請參閱 取得 Amazon Rekognition Video 分析結果。

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
    "MaxResults": 10,
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw=="
}
```

GetSegmentDetection 會傳回要求的分析結果，以及關於存放影片的一般資訊。

一般資訊

GetSegmentDection 會傳回以下一般資訊。

- 音訊資訊 — 回應包含AudioMetadata物件陣列中的音訊中繼資料。AudioMetadata可以有多個音訊串流。每個 AudioMetadata 物件都包含單一音訊串流的中繼資料。AudioMetadata 物件中的音訊資訊包括音訊轉碼器、音訊聲道的數目、音訊串流的持續時間，以及取樣率。GetSegmentDetection 傳回的每一頁資訊中都會傳回音訊中繼資料。
- 影片資訊 — 目前，Amazon Rekognition Video 訊會傳回陣列中的單一VideoMetadata物件。VideoMetadata物件包含 Amazon Rekognition Video 選擇要分析之輸入檔案中影片串流的相關資訊。VideoMetadata 物件包括影片轉碼器、影片格式，以及其他資訊。GetSegmentDetection 傳回的每一頁資訊中都會傳回影片中繼資料。
- 分頁資訊：此範例顯示單頁的區段資訊。您可以在 GetSegmentDetection 的 MaxResults 輸入參數中指定要傳回的元素數目。如果結果數目超過 MaxResults，GetSegmentDetection 會傳回

用來取得下一頁結果的字符 (NextToken)。如需詳細資訊，請參閱 取得 Amazon Rekognition Video 分析結果。

- 要求資訊：呼叫 StartSegmentDetection 時要求的分析類型會在 SelectedSegmentTypes 欄位中傳回。

客群

在視訊中偵測到的技術線索和拍攝資訊會以SegmentDetection物件的陣列傳回。Segments陣列會依照 StartSegmentDetection 之 SegmentTypes 輸入參數中指定的區段類型 (技術提示或鏡頭) 排序。在每個區段類型中，陣列會依照時間戳記值排序。每個 SegmentDetection 物件都包含偵測到的區段類型 (技術提示或鏡頭偵測) 相關資訊，以及一般資訊，例如，開始時間、結束時間，以及區段的持續時間。

時間資訊會以兩種格式傳回。

- 毫秒

  影片開始後的毫秒數。欄位 DurationMillis、StartTimestampMillis 和 EndTimestampMillis 的格式為毫秒。
- 時間碼

  Amazon Rekognition Video 時間碼採用 SMPTE 格式，其中每個影片影格都有唯一的時間碼值。格式為 hh:mm:ss:frame。例如，時間碼值 01:05:40:07 讀為一小時、五分鐘、四十秒和七個影格。Amazon Rekognition Video影片支援丟棄畫面播放速率使用案例。丟棄速率時間碼格式為 hh:mm:ss;frame。欄位 DurationSMPTE、StartTimecodeSMPTE 和 EndTimecodeSMPTE 為時間碼格式。
- 影格計數器

  每個影片片段的持續時間也會以畫面數目來表示。此欄位 StartFrameNumber 會在影片片段的開頭 EndFrameNumber 在影片片段的結尾提供影格編號。DurationFrames 表示影片區段中的影格總數。這些值是使用以 0 開頭的影格索引來計算。

您可以使用 SegmentType 欄位來決定 Amazon Rekognition Video 傳回的區段類型。

- 技術提示 — TechnicalCueSegment 欄位是包含偵測信賴度和技術提示類型的TechnicalCueSegment物件。技術提示的類型為 ColorBars、EndCredits、BlackFrames、OpeningCredits、StudioLogo、Slate、和 Content。

- 鏡頭 — ShotSegment 欄位是包含偵測信賴度和視訊中快照區段識別碼的ShotSegment物件。

以下是來自 GetSegmentDetection 的 JSON 回應範例。

```
{
    "SelectedSegmentTypes": [
        {
            "ModelVersion": "2.0",
            "Type": "SHOT"
        },
        {
            "ModelVersion": "2.0",
            "Type": "TECHNICAL_CUE"
        }
    ],
    "Segments": [
        {
            "DurationFrames": 299,
            "DurationSMPTE": "00:00:09;29",
            "StartFrameNumber": 0,
            "EndFrameNumber": 299,
            "EndTimecodeSMPTE": "00:00:09;29",
            "EndTimestampMillis": 9976,
            "StartTimestampMillis": 0,
            "DurationMillis": 9976,
            "StartTimecodeSMPTE": "00:00:00;00",
            "Type": "TECHNICAL_CUE",
            "TechnicalCueSegment": {
                "Confidence": 90.45006561279297,
                "Type": "BlackFrames"
            }
        },
        {
            "DurationFrames": 150,
            "DurationSMPTE": "00:00:05;00",
            "StartFrameNumber": 299,
            "EndFrameNumber": 449,
            "EndTimecodeSMPTE": "00:00:14;29",
            "EndTimestampMillis": 14981,
            "StartTimestampMillis": 9976,
            "DurationMillis": 5005,
            "StartTimecodeSMPTE": "00:00:09;29",
            "Type": "TECHNICAL_CUE",
```

```
            "TechnicalCueSegment": {
                "Confidence": 100.0,
                "Type": "Content"
            }
        },
        {
            "DurationFrames": 299,
            "ShotSegment": {
                "Index": 0,
                "Confidence": 99.9982681274414
            },
            "DurationSMPTE": "00:00:09;29",
            "StartFrameNumber": 0,
            "EndFrameNumber": 299,
            "EndTimecodeSMPTE": "00:00:09;29",
            "EndTimestampMillis": 9976,
            "StartTimestampMillis": 0,
            "DurationMillis": 9976,
            "StartTimecodeSMPTE": "00:00:00;00",
            "Type": "SHOT"
        },
        {
            "DurationFrames": 149,
            "ShotSegment": {
                "Index": 1,
                "Confidence": 99.9982681274414
            },
            "DurationSMPTE": "00:00:04;29",
            "StartFrameNumber": 300,
            "EndFrameNumber": 449,
            "EndTimecodeSMPTE": "00:00:14;29",
            "EndTimestampMillis": 14981,
            "StartTimestampMillis": 10010,
            "DurationMillis": 4971,
            "StartTimecodeSMPTE": "00:00:10;00",
            "Type": "SHOT"
        }
    ],
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": [
        {
            "Format": "QuickTime / MOV",
            "FrameRate": 29.970029830932617,
            "Codec": "h264",
```

```
            "DurationMillis": 15015,
            "FrameHeight": 1080,
            "FrameWidth": 1920,
            "ColorRange": "LIMITED"

        }
    ],
    "AudioMetadata": [
        {
            "NumberOfChannels": 1,
            "SampleRate": 48000,
            "Codec": "aac",
            "DurationMillis": 15007
        }
    ]
}
```

如需範例程式碼，請參閱 範例：偵測已存放影片中的區段。

範例：偵測已存放影片中的區段

下列程序顯示如何偵測存放在 Amazon S3 儲存貯體之影片中的技術提示區段和鏡頭偵測區段。此程序也會示範如何根據 Amazon Rekognition Video 對於偵測準確性的可信度，篩選偵測到的區段。

此範例會展開 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 中的程式碼，這會使用 Amazon Simple Queue Service 佇列來取得影片分析要求的完成狀態。

偵測存放在 Amazon S3 儲存貯體 (SDK) 之影片中的區段

1. 執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片。
2. 將下列專案新增至您在步驟 1 中使用的程式碼。

   Java

   1. 新增以下匯入專案。

```
import com.amazonaws.services.rekognition.model.GetSegmentDetectionRequest;
import com.amazonaws.services.rekognition.model.GetSegmentDetectionResult;
import com.amazonaws.services.rekognition.model.SegmentDetection;
import com.amazonaws.services.rekognition.model.SegmentType;
import com.amazonaws.services.rekognition.model.SegmentTypeInfo;
import com.amazonaws.services.rekognition.model.ShotSegment;
```

```
import
 com.amazonaws.services.rekognition.model.StartSegmentDetectionFilters;
import
 com.amazonaws.services.rekognition.model.StartSegmentDetectionRequest;
import com.amazonaws.services.rekognition.model.StartSegmentDetectionResult;
import com.amazonaws.services.rekognition.model.StartShotDetectionFilter;
import
 com.amazonaws.services.rekognition.model.StartTechnicalCueDetectionFilter;
import com.amazonaws.services.rekognition.model.TechnicalCueSegment;
import com.amazonaws.services.rekognition.model.AudioMetadata;
```

2. 將下列程式碼新增至類別 VideoDetect。

```
    //Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights
 Reserved.
    //PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)


    private static void StartSegmentDetection(String bucket, String video)
 throws Exception{

        NotificationChannel channel= new NotificationChannel()
                .withSNSTopicArn(snsTopicArn)
                .withRoleArn(roleArn);

        float minTechnicalCueConfidence = 80F;
        float minShotConfidence = 80F;

        StartSegmentDetectionRequest req = new
 StartSegmentDetectionRequest()
                .withVideo(new Video()
                        .withS3Object(new S3Object()
                                .withBucket(bucket)
                                .withName(video)))
                .withSegmentTypes("TECHNICAL_CUE" , "SHOT")
                .withFilters(new StartSegmentDetectionFilters()
                        .withTechnicalCueFilter(new
 StartTechnicalCueDetectionFilter()

 .withMinSegmentConfidence(minTechnicalCueConfidence))
                        .withShotFilter(new StartShotDetectionFilter()

 .withMinSegmentConfidence(minShotConfidence)))
```

```
                .withJobTag("DetectingVideoSegments")
                .withNotificationChannel(channel);

        StartSegmentDetectionResult startLabelDetectionResult =
 rek.startSegmentDetection(req);
        startJobId=startLabelDetectionResult.getJobId();

    }

    private static void GetSegmentDetectionResults() throws Exception{

        int maxResults=10;
        String paginationToken=null;
        GetSegmentDetectionResult segmentDetectionResult=null;
        Boolean firstTime=true;


        do {
            if (segmentDetectionResult !=null){
                paginationToken = segmentDetectionResult.getNextToken();
            }

            GetSegmentDetectionRequest segmentDetectionRequest= new
 GetSegmentDetectionRequest()
                    .withJobId(startJobId)
                    .withMaxResults(maxResults)
                    .withNextToken(paginationToken);

            segmentDetectionResult =
 rek.getSegmentDetection(segmentDetectionRequest);

            if(firstTime) {
                System.out.println("\nStatus\n------");
                System.out.println(segmentDetectionResult.getJobStatus());
                System.out.println("\nRequested features
\n------------------");
                 for (SegmentTypeInfo requestedFeatures :
 segmentDetectionResult.getSelectedSegmentTypes()) {
                    System.out.println(requestedFeatures.getType());
                }
                 int count=1;
                 List<VideoMetadata> videoMetaDataList =
 segmentDetectionResult.getVideoMetadata();
                 System.out.println("\nVideo Streams\n-------------");
```

```
                for (VideoMetadata videoMetaData: videoMetaDataList) {
                    System.out.println("Stream: " + count++);
                    System.out.println("\tFormat: " +
videoMetaData.getFormat());
                    System.out.println("\tCodec: " +
videoMetaData.getCodec());
                    System.out.println("\tDuration: " +
videoMetaData.getDurationMillis());
                    System.out.println("\tFrameRate: " +
videoMetaData.getFrameRate());
                }


                List<AudioMetadata> audioMetaDataList =
segmentDetectionResult.getAudioMetadata();
                System.out.println("\nAudio streams\n-------------");

                count=1;
                for (AudioMetadata audioMetaData: audioMetaDataList) {
                    System.out.println("Stream: " + count++);
                    System.out.println("\tSample Rate: " +
audioMetaData.getSampleRate());
                    System.out.println("\tCodec: " +
audioMetaData.getCodec());
                    System.out.println("\tDuration: " +
audioMetaData.getDurationMillis());
                    System.out.println("\tNumber of Channels: " +
audioMetaData.getNumberOfChannels());
                }
                System.out.println("\nSegments\n--------");

               firstTime=false;
           }


           //Show segment information

           List<SegmentDetection> detectedSegments=
segmentDetectionResult.getSegments();

           for (SegmentDetection detectedSegment: detectedSegments) {
```

```
                if
(detectedSegment.getType().contains(SegmentType.TECHNICAL_CUE.toString()))
{
                    System.out.println("Technical Cue");
                    TechnicalCueSegment
segmentCue=detectedSegment.getTechnicalCueSegment();
                    System.out.println("\tType: " + segmentCue.getType());
                    System.out.println("\tConfidence: " +
segmentCue.getConfidence().toString());
                }
                if
(detectedSegment.getType().contains(SegmentType.SHOT.toString())) {
                    System.out.println("Shot");
                    ShotSegment
segmentShot=detectedSegment.getShotSegment();
                    System.out.println("\tIndex " +
segmentShot.getIndex());
                    System.out.println("\tConfidence: " +
segmentShot.getConfidence().toString());
                }
                long seconds=detectedSegment.getDurationMillis();
                System.out.println("\tDuration : " + Long.toString(seconds)
+ " milliseconds");
                System.out.println("\tStart time code: " +
detectedSegment.getStartTimecodeSMPTE());
                System.out.println("\tEnd time code: " +
detectedSegment.getEndTimecodeSMPTE());
                System.out.println("\tDuration time code: " +
detectedSegment.getDurationSMPTE());
                System.out.println();

            }

        } while (segmentDetectionResult !=null &&
segmentDetectionResult.getNextToken() != null);

    }
```

3. 在函數 main 中，將下行:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
```

```
         GetLabelDetectionResults();
```

取代為：

```
        StartSegmentDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetSegmentDetectionResults();
```

Java V2

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectVideoSegments {
```

```
private static String startJobId ="";
public static void main(String[] args) {

    final String usage = "\n" +
        "Usage: " +
        "   <bucket> <video> <topicArn> <roleArn>\n\n" +
        "Where:\n" +
        "   bucket - The name of the bucket in which the video is located (for
example, (for example, myBucket). \n\n"+
        "   video - The name of video (for example, people.mp4). \n\n" +
        "   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic. \n\n" +
        "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
role to use. \n\n" ;

    if (args.length != 4) {
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_WEST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    startTextLabels(rekClient, channel, bucket, video);
    GetTextResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_text.main]
```

```
public static void startTextLabels(RekognitionClient rekClient,
                                NotificationChannel channel,
                                String bucket,
                                String video) {
    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();

        Video vidOb = Video.builder()
            .s3Object(s3Obj)
            .build();

        StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
            .jobTag("DetectingLabels")
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
        startJobId = labelDetectionResponse.jobId();

    } catch (RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}

public static void GetTextResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetTextDetectionResponse textDetectionResponse=null;
        boolean finished = false;
        String status;
        int yy=0 ;

        do{
            if (textDetectionResponse !=null)
                paginationToken = textDetectionResponse.nextToken();
```

```
            GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                .jobId(startJobId)
                .nextToken(paginationToken)
                .maxResults(10)
                .build();

            // Wait until the job succeeds.
            while (!finished) {
                textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                status = textDetectionResponse.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
                    Thread.sleep(1000);
                }
                yy++;
            }

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null.
            VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<TextDetectionResult> labels=
textDetectionResponse.textDetections();
            for (TextDetectionResult detectedText: labels) {
                System.out.println("Confidence: " +
detectedText.textDetection().confidence().toString());
                System.out.println("Id : " +
detectedText.textDetection().id());
                System.out.println("Parent Id: " +
detectedText.textDetection().parentId());
                System.out.println("Type: " +
detectedText.textDetection().type());
```

```
                System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                System.out.println();
            }

        } while (textDetectionResponse !=null &&
 textDetectionResponse.nextToken() != null);

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_text.main]
}
```

Python

1. 將下列程式碼新增至您在步驟 1 中建立的類別 VideoDetect。

```
# Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def StartSegmentDetection(self):

        min_Technical_Cue_Confidence = 80.0
        min_Shot_Confidence = 80.0
        max_pixel_threshold = 0.1
        min_coverage_percentage = 60

        response = self.rek.start_segment_detection(
            Video={"S3Object": {"Bucket": self.bucket, "Name": self.video}},
            NotificationChannel={
                "RoleArn": self.roleArn,
                "SNSTopicArn": self.snsTopicArn,
            },
            SegmentTypes=["TECHNICAL_CUE", "SHOT"],
            Filters={
                "TechnicalCueFilter": {
                    "BlackFrame": {
                        "MaxPixelThreshold": max_pixel_threshold,
                        "MinCoveragePercentage": min_coverage_percentage,
```

```
                    },
                    "MinSegmentConfidence": min_Technical_Cue_Confidence,
                },
                "ShotFilter": {"MinSegmentConfidence": min_Shot_Confidence},
            }
        )

        self.startJobId = response["JobId"]
        print(f"Start Job Id: {self.startJobId}")

    def GetSegmentDetectionResults(self):
        maxResults = 10
        paginationToken = ""
        finished = False
        firstTime = True

        while finished == False:
            response = self.rek.get_segment_detection(
                JobId=self.startJobId, MaxResults=maxResults,
NextToken=paginationToken
            )

            if firstTime == True:
                print(f"Status\n------\n{response['JobStatus']}")
                print("\nRequested Types\n---------------")
                for selectedSegmentType in response['SelectedSegmentTypes']:
                    print(f"\tType: {selectedSegmentType['Type']}")
                    print(f"\t\tModel Version:
{selectedSegmentType['ModelVersion']}")

                print()
                print("\nAudio metadata\n--------------")
                for audioMetadata in response['AudioMetadata']:
                    print(f"\tCodec: {audioMetadata['Codec']}")
                    print(f"\tDuration: {audioMetadata['DurationMillis']}")
                    print(f"\tNumber of Channels:
{audioMetadata['NumberOfChannels']}")
                    print(f"\tSample rate: {audioMetadata['SampleRate']}")
                print()
                print("\nVideo metadata\n--------------")
                for videoMetadata in response["VideoMetadata"]:
                    print(f"\tCodec: {videoMetadata['Codec']}")
                    print(f"\tColor Range: {videoMetadata['ColorRange']}")
                    print(f"\tDuration: {videoMetadata['DurationMillis']}")
```

```
                    print(f"\tFormat: {videoMetadata['Format']}")
                    print(f"\tFrame rate: {videoMetadata['FrameRate']}")
                    print("\nSegments\n--------")

                firstTime = False

            for segment in response['Segments']:

                if segment["Type"] == "TECHNICAL_CUE":
                    print("Technical Cue")
                    print(f"\tConfidence: {segment['TechnicalCueSegment']
['Confidence']}")
                    print(f"\tType: {segment['TechnicalCueSegment']
['Type']}")

                if segment["Type"] == "SHOT":
                    print("Shot")
                    print(f"\tConfidence: {segment['ShotSegment']
['Confidence']}")
                    print(f"\tIndex: " + str(segment["ShotSegment"]
["Index"]))

                print(f"\tDuration (milliseconds):
 {segment['DurationMillis']}")
                print(f"\tStart Timestamp (milliseconds):
 {segment['StartTimestampMillis']}")
                print(f"\tEnd Timestamp (milliseconds):
 {segment['EndTimestampMillis']}")

                print(f"\tStart timecode: {segment['StartTimecodeSMPTE']}")
                print(f"\tEnd timecode: {segment['EndTimecodeSMPTE']}")
                print(f"\tDuration timecode: {segment['DurationSMPTE']}")

                print(f"\tStart frame number {segment['StartFrameNumber']}")
                print(f"\tEnd frame number: {segment['EndFrameNumber']}")
                print(f"\tDuration frames: {segment['DurationFrames']}")

                print()

            if "NextToken" in response:
                paginationToken = response["NextToken"]
            else:
                finished = True
```

2. 在函數 main 中，將下行:

```
analyzer.StartLabelDetection()
if analyzer.GetSQSMessageSuccess()==True:
    analyzer.GetLabelDetectionResults()
```

取代為：

```
analyzer.StartSegmentDetection()
if analyzer.GetSQSMessageSuccess()==True:
    analyzer.GetSegmentDetectionResults()
```

> **Note**
>
> 如果您已執行 使用 Java 或 Python (SDK) 分析儲存於 Amazon S3 儲存貯體中的影片 以外的影片範例，要取代的程式碼可能會不同。

3. 執行程式碼。隨即顯示在輸入影片中偵測到的區段相關資訊。

偵測人臉活體

Amazon Rekognition 人臉活體可協助您確認進行人臉驗證的使用者是否實際位於攝影機前方。其可以偵測攝影機面臨的欺騙攻擊或試圖繞過攝影機的操作。使用者可以通過拍攝短影片自拍來完成人臉活體檢查，並按照一系列旨在驗證其存在的提示進行操作。

人臉活體是透過機率計算來決定，然後在檢查之後傳回可信度分數 (介於 0 至 100 之間)。得分越高，接受檢查的人的可信度就越大。人臉活體還傳回一個畫面，稱為可用於人臉比較和搜尋的參考映像。與任何基於概率的系統一樣，人臉活體無法保證完美的結果。將其與其他因素一起使用，對使用者的個人身份做出基於風險的決定。

人臉活體使用多種組成部分：

- AWS Amplify SDK (反應, 斯威夫特 (iOS), 和安卓系統) 與 FaceLivenessDetector 組件
- AWS 開發套件
- AWS 雲端 API

當您設定應用程式與人臉活體特徵整合時，使用以下 API 操作：

- CreateFaceLivenessSession-啟動臉部活力會話，讓臉部活力檢測模型在您的應用程序中使用。返回一 SessionId 個創建的會話。
- StartFaceLivenessSession-被 AWS Amplify FaceLivenessDetector 調用。啟動事件串流，其中包含目前工作階段中相關事件和屬性的相關資訊。
- GetFaceLivenessSession結果-擷取特定「臉部活力」工作階段的結果，包括「臉部活力」信賴度分數、參考影像和稽核影像。

您將使用 Ampli AWS fy SDK 將臉部活力功能與 Web 應用程式的臉部基礎驗證工作流程整合在一起。當使用者透過您的應用程式上線或驗證時，請將使用者傳送至 Amplify SDK 中的人臉活體檢查工作流程。Amplify SDK 會在使用者進行影片自拍時為使用者處理使用者介面和即時回饋。

當使用者的人臉移至裝置上顯示的橢圓形時，Amplify SDK 會在螢幕上顯示一系列彩色燈光。然後，它將自拍影片安全地串流至雲端 API。雲端 API 使用進階機器學習模型進行即時分析。分析完成後，您會在後端收到以下內容：

- 人臉活體信心得分 (介於 0 到 100 之間)
- 稱為參考映像的高質量映像可用於人臉匹配或人臉搜尋

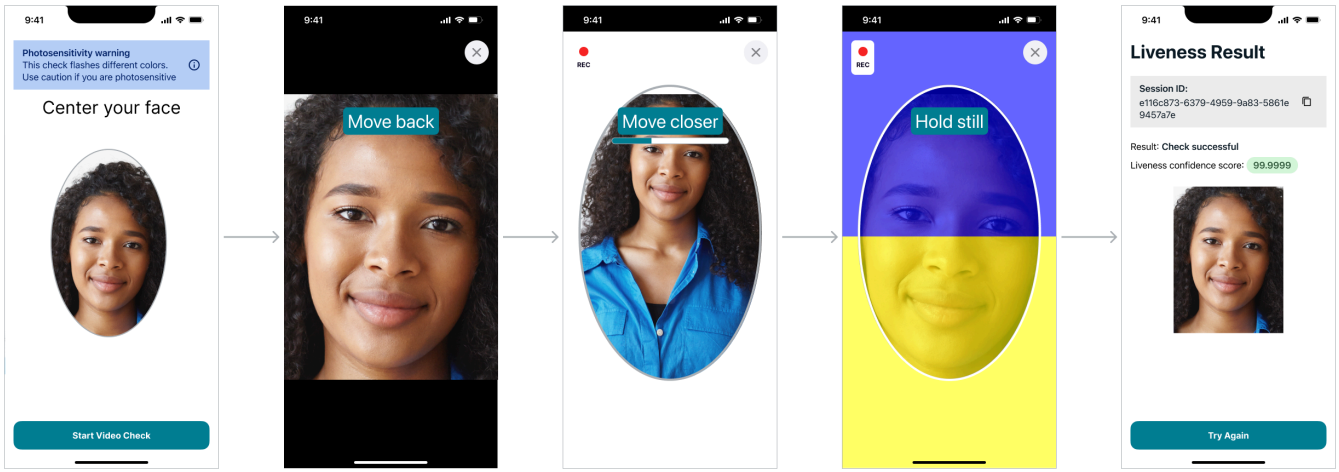
- 一組最多四張影像稱為稽核影像，從自拍影片中選取

人臉活體可用於各種用例。例如，Face Liveness 可以與面部匹配（CompareFaces和SearchFacesByImage）一起使用，用於身份驗證，用於具有基於年齡訪問限制的平台上進行年齡估算，以及在阻止機器人時檢測真實的人類用戶。

您可以進一步了解服務的目的使用案例、服務如何使用機器學習 (ML)，以及 Rekognition 人臉活體 AI 服務卡中負責任設計和使用服務的主要考量事項。

您可以設定人臉活體和人臉比對可信度分數的閾值。您選擇的閾值應該反映您的使用案例。然後，您會根據分數高於或低於閾值，向使用者傳送身分驗證核准/拒絕。如果被拒絕，要求使用者再試一次，或將其傳送至其他方法。

下圖展示了使用者流程，從指令到活性檢查到傳回結果：



使用者端人臉活體要求

Amazon Rekognition 人臉活體需要以下最低規格：

裝置：

- 裝置必須具有前置攝影機
- 裝置顯示屏的最小刷新率：60 赫茲
- 最小顯示器或熒幕尺寸：4 英寸
- 裝置不應被破解或獲得最高系統權限

相機規格：

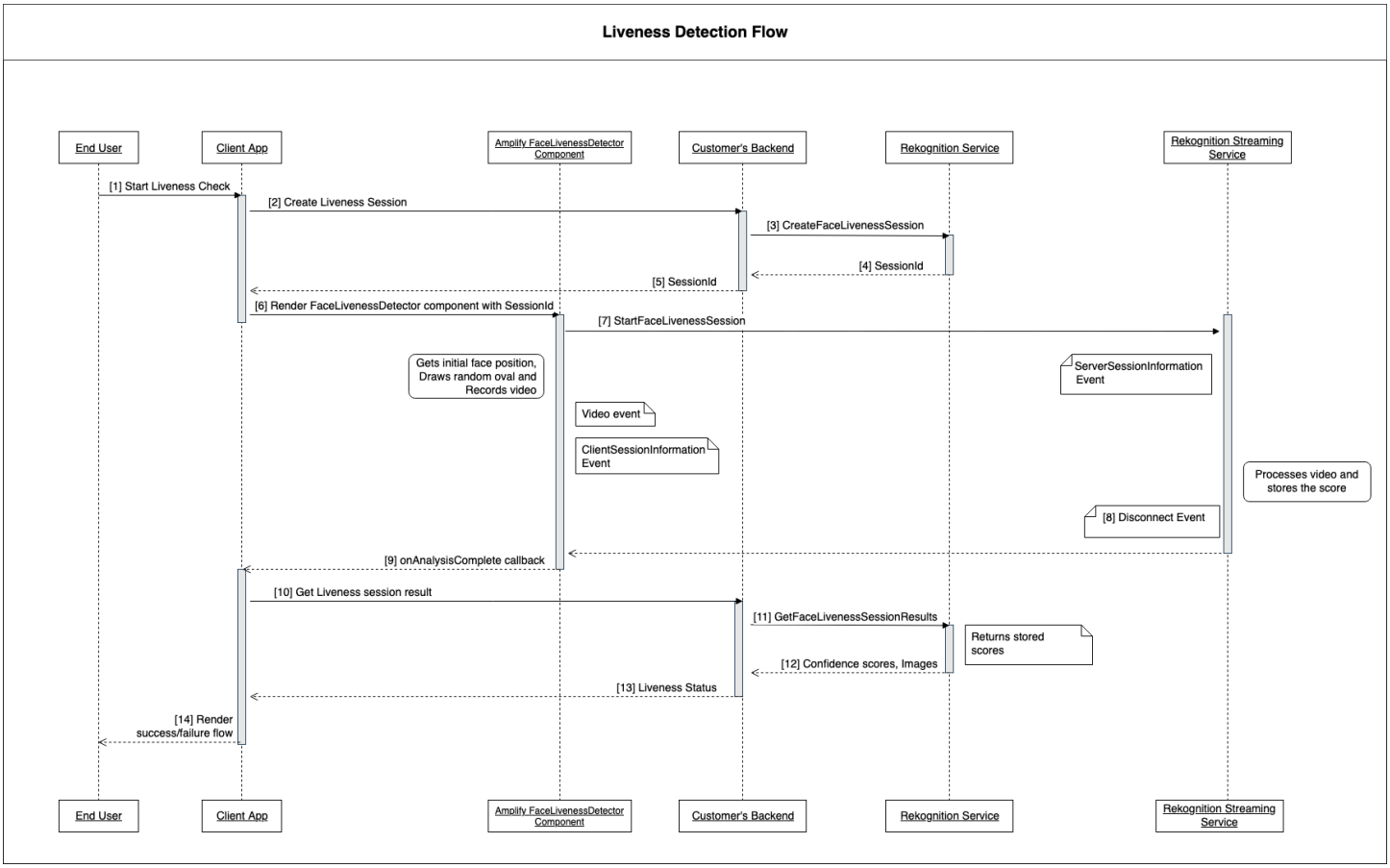
- 彩色照相機：前置攝影機應該能夠記錄彩色。
- 沒有虛擬相機或相機軟件。
- 最低錄音能力：每秒 15 幀。
- 最低影片錄製分辨率：320 x 240 像素。
- 當使用者使用帶有桌面的網路攝影機進行人臉活體檢查時，重要的是將網路攝影機安裝於開始人臉活體檢查的同一熒幕上。

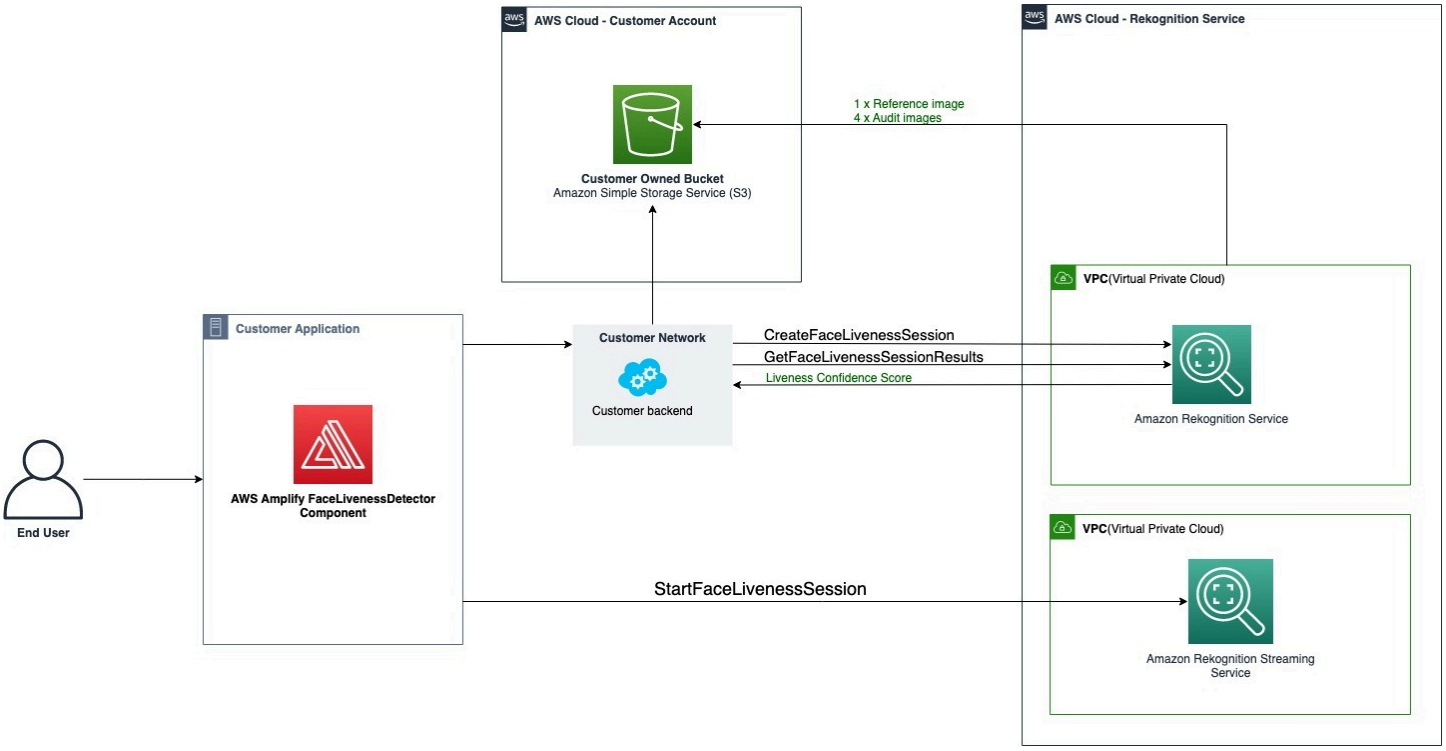
最低頻寬要求：100 千比特

支援的瀏覽器：最新三個版本的主流瀏覽器，如 Google Chrome、Mozilla Firefox、Apple Safari 和 Microsoft Edge。如需瀏覽器支援的更多資訊，請參閱什麼瀏覽器可搭配 AWS 管理主控台使用？

架構和序列圖

下圖詳細說明了 Amazon Rekognition 人臉活體在該特徵的架構和操作順序方面的運作方式：





人臉活性檢查程序包含下列幾個步驟：

1. 使用者在使用者端應用程式中啟動人臉活體檢查。
2. 使用者端應用程式會呼叫客戶的後端，進而呼叫 Amazon Rekognition 服務。該服務創建一個面部活力會話，並返回一個唯 SessionId一的。注意：發送後，它會在 3 分鐘內過期，因此只有 3 分鐘的窗口可以完成下面的步驟 3 到 7。 SessionId 每次臉部活力檢查都必須使用新的工作階段 ID。如果給定的 SessionId 用於後續的面部活力檢查，則檢查將失敗。此外，會在傳送 3 分鐘後 SessionId 過期，使得與工作階段相關聯的所有 Liveness 資料 (例如 SessionId、參考映像、稽核映像等) 無法使用。
3. 用戶端應用程式會使用取得的 SessionId 和適當的回呼來呈現 FaceLivenessDetector Amplify 元件。
4. FaceLivenessDetector 元件會建立與 Amazon Rekognition 串流服務的連線、在使用者的螢幕上呈現橢圓形，並顯示一系列彩色燈號。 FaceLivenessDetector 將視訊即時記錄並串流至 Amazon Rekognition 串流服務。
5. Amazon Rekognition 串流服務會即時處理視訊、儲存結果，並在串流完 FaceLivenessDetector 成時傳回元件。 DisconnectEvent
6. FaceLivenessDetector 元件會呼叫`onAnalysisComplete`回呼，向用戶端應用程式發出訊號，表示串流已完成，且分數已準備好可供擷取。

7. 客戶端應用程式調用客戶的後端以獲取一個布爾標誌，表明使用者是否在線。客戶後端向 Amazon Rekognition 服務提出要求，以取得可信度分數、參考和稽核映像。客戶後端使用這些屬性來確定使用者是否處於活動狀態，並向客戶端應用程式傳回適當的回應。
8. 最後，用戶端應用程式會將回應傳遞給 FaceLivenessDetector元件，該元件會適當呈現成功/失敗訊息以完成流程。

必要條件

使用 Amazon Rekognition 人臉活體的先決條件包括以下內容：

1. 設置一個 AWS 帳戶
2. 設定臉部活力 SDK AWS
3. 設定 AWS Amplify 資源

步驟 1：設定 AWS 帳戶

如果您還沒有 AWS 帳號，請完成所述的步驟創建一個 AWS 帳戶和用戶以建立帳戶。

步驟 2：設定人臉活體 AWS SDK

如果您尚未安裝，請安裝 AWS CLI 並設定和 AWS SDK。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

有幾種方法可以驗證 AWS SDK 呼叫。本指南中的範例假設您使用預設認證設定檔來呼叫 AWS CLI 命令和 AWS SDK API 作業。

請參閱授與程式設計存取權頁面，以取得有關授與使用者帳戶存取您所選 AWS SDK 的詳細資訊。此頁面也說明如何在本機電腦上使用設定檔，以及如何在 AWS 環境中執行範例程式碼。

請確定呼叫人臉活體作業的使用者具有呼叫作業的正確權限，例如 `AmazonRekognitionFullAccess` 和 `AmazonS3FullAccess` 權限。

步驟 3：設定 AWS Amplify 資源

若要將 Amazon Rekognition 臉部活力整合到您的應用程式中，您必須設定 Amplify 開發套件以使用 Ampli AWS fy 元件。 FaceLivenessDetector

如果您還沒有，請按照說明設定 AWS Command Line Interface (AWS CLI)入門時設定 AWS CLI。安裝 CLI 之後，請完成 A mplify UI 文件網站上的 [設定驗證] 步驟，以設定您的 A AWS mplify 資源。

偵測人臉活體的最佳實務

我們建議您在使用 Amazon Rekognition 人臉活體時，遵循數種最佳實務。人臉活體的最佳實務包括應在何處進行人臉活體檢查、使用稽核映像，以及選擇可信度分數閾值的準則。

如需最佳作法的完整清單，請參閱 人臉活體的使用建議。

編程 Amazon Rekognition 人臉活體 API

若要使用 Amazon Rekognition 人臉活體 API，您必須建立執行下列步驟的後端：

1. 呼CreateFaceLivenessSession籲啟動臉部活力會話。`CreateFaceLivenessSession` 操作完成後，UI 會提示使用者提交影片自拍照。然後，AWS 擴增的 FaceLivenessDetector 元件會呼叫StartFacelivenessSession以執行活性偵測。
2. 呼叫GetFaceLivenessSession結果以傳回與「臉部活力」工作階段相關聯的偵測結果。
3. 按照 Amp lify 生命指南中的步驟，繼續配置您的 React 應用程序以使用該 FaceLivenessDetector組件。

使用人臉活體之前，請確定您已建立 AWS 帳戶、設定 AWS CLI 和 AWS SDK，以及設定 AWS Amplify。您還應確保後端 API 的 IAM 政策具有涵蓋以下內容的權限：`GetFaceLivenessSessionResults` 和 `CreateFaceLivenessSession`。如需詳細資訊，請參閱 必要條件 一節。

步驟 1： CreateFaceLivenessSession

CreateFaceLivenessSession API 操作創建一個面部活性會話，並返回一個唯`SessionId`一的。

作為此操作輸入的一部分，也可以指定 Amazon S3 儲存貯體位置。這允許儲存參考映像並審核在人臉活體工作階段期間生成的映像。Amazon S3 儲存貯體必須在呼叫者的 AWS 帳戶中，與人臉活體端點位於同一區域。此外，S3 物件金鑰是由人臉活體系統所產生。

也可以提供 `AuditImagesLimit`，其是一個介於 0 和 4 之間的數字。預設會設定為 0。傳回的映像數量是盡最大努力的結果，並以自拍影片的持續時間為基礎。

請求範例

```
{
    "ClientRequestToken": "my_default_session",
    "Settings": {
```

```
        "OutputConfig": {
            "S3Bucket": "s3bucket",
            "S3KeyPrefix": "s3prefix"
        },
        "AuditImagesLimit": 1
    }
}
```

回應範例

```
{
    {"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
}
```

步驟 2： StartFaceLivenessSession

CreateFaceLivenessSession API 作業完成時，AWS Amplify 元件會執行 StartFaceLivenessSession API 作業。系統會提示使用者擷取影片自拍照。為了進行成功的檢查，使用者必須將人臉放置於熒幕上的橢圓形內，同時保持良好的照明。如需詳細資訊，請參閱 人臉活體的使用建議。

此 API 作業需要臉部生命性工作階段期間擷取的視訊、從 CreateFaceLivenessSession API 作業取得的 sessionId，以及回呼。onAnalysisComplete回呼可用於向後端發出訊號，以呼叫 GetFaceLivenessSessionResults API 作業，該作業會傳回可信度分數、參考和稽核影像。

請注意，此步驟是由用戶端應用程式上的 AWS Amplify FaceLivenessDetector 元件執行。您不需要執行其他設定即可呼叫 StartFaceLivenessSession。

步驟 3： GetFaceLivenessSessionResults

GetFaceLivenessSessionResults API 作業會擷取特定臉部活力工作階段的結果。該操作需要輸入 sessionId，並傳回相應的人臉活性可信度分數。其還提供包括人臉邊界框的參考映像，以及檢核也包含人臉邊界框的映像。人臉活體的可信度得分範圍為 0 至 100。

請求範例

```
{"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
```

回應範例

```
{
    "SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8",
    "Confidence": 98.9735,
    "ReferenceImage": {
        "S3Object": {
            "Bucket": "s3-bucket-name",
            "Name": "file-name",
        },
        "BoundingBox": {
           "Height": 0.4943420886993408,
            "Left": 0.8435328006744385,
            "Top": 0.8435328006744385,
            "Width": 0.9521094560623169}
    },
    "AuditImages": [{
        "S3Object": {
            "Bucket": "s3-bucket-name",
            "Name": "audit-image-name",
        },
        "BoundingBox": {
           "Width": 0.6399999856948853,
           "Height": 0.47999998927116394,
           "Left": 0.1644444465637207,
           "Top": 0.17666666209697723}
    }],
    "Status": "SUCCEEDED"
}
```


步驟 4：回應結果

在人臉活體工作階段之後，將檢查的可信度分數與指定的臨界值進行比較。如果分數高於閾值，則使用者可以轉到下一個螢幕或任務。如果檢查失敗，系統會通知使用者並提示您再試一次。

呼叫人臉活體 API

您可以使用任何支援的開發套件 (例如 AWS Python 開發套件 Boto3 或適用 AWS AWS SDK for Java 的 AWS 開發套件) 來測試亞馬遜重新認知臉部活力。您可以使用所選的 SDK 來呼叫 `CreateFaceLivenessSession` 和 `GetFaceLivenessSessionResults` API。下一節將示範如何使用 Python 和 Java SDK 呼叫這些 API。

若要呼叫人臉活體 API：

- 如果您尚未這麼做，請建立或更新具有 AmazonRekognitionFullAccess 權限的使用者。如需詳細資訊，請參閱步驟 1：設定 AWS 帳戶並建立使用者。
- 如果您尚未準備就緒，請安裝並設定 AWS CLI 與 AWS SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和 AWS SDK。

Python

下面的片段顯示了如何在 Python 應用程式中呼叫這些 API。請注意，若要執行此範例，您必須至少使用 Boto3 SDK 的 1.26.110 版本，但建議您使用最新版本的 SDK。

```
import boto3

session = boto3.Session(profile_name='default')
client = session.client('rekognition')

def create_session():

    response = client.create_face_liveness_session()

    session_id = response.get("SessionId")
    print('SessionId: ' + session_id)

    return session_id


def get_session_results(session_id):

    response = client.get_face_liveness_session_results(SessionId=session_id)

    confidence = response.get("Confidence")
    status = response.get("Status")

    print('Confidence: ' + "{:.2f}".format(confidence) + "%")
    print('Status: ' + status)

    return status


def main():
```

```
    session_id = create_session()
    print('Created a Face Liveness Session with ID: ' + session_id)

    status = get_session_results(session_id)
    print('Status of Face Liveness Session: ' + status)


if __name__ == "__main__":
    main()
```

Java

下列程式碼片段顯示如何在 Java 應用程式中呼叫這些 API：

```
package aws.example.rekognition.liveness;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionRequest;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionResult;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsRequest;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsResult;

public class DemoLivenessApplication {

    static AmazonRekognition rekognitionClient;

    public static void main(String[] args) throws Exception {

        rekognitionClient = AmazonRekognitionClientBuilder.defaultClient();

        try {
            String sessionId = createSession();
            System.out.println("Created a Face Liveness Session with ID: " +
 sessionId);

            String status = getSessionResults(sessionId);
            System.out.println("Status of Face Liveness Session: " + status);
```

```
        } catch(AmazonRekognitionException e) {
           e.printStackTrace();
        }
    }

    private static String createSession() throws Exception {

        CreateFaceLivenessSessionRequest request = new
 CreateFaceLivenessSessionRequest();
        CreateFaceLivenessSessionResult result =
 rekognitionClient.createFaceLivenessSession(request);

        String sessionId = result.getSessionId();
        System.out.println("SessionId: " + sessionId);

        return sessionId;
    }

    private static String getSessionResults(String sessionId) throws Exception {

        GetFaceLivenessSessionResultsRequest request = new
 GetFaceLivenessSessionResultsRequest().withSessionId(sessionId);
        GetFaceLivenessSessionResultsResult result =
 rekognitionClient.getFaceLivenessSessionResults(request);

        Float confidence = result.getConfidence();
        String status = result.getStatus();

        System.out.println("Confidence: " + confidence);
        System.out.println("status: " + status);

        return status;
    }
}
```

Java V2

下列程式碼片段示範如何使用 AWS Java V2 SDK 呼叫臉部活力 API：

```
package aws.example.rekognition.liveness;
```

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionRequest;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionResult;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsRequest;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsResult;

public class DemoLivenessApplication {

    static AmazonRekognition rekognitionClient;

    public static void main(String[] args) throws Exception {

        rekognitionClient = AmazonRekognitionClientBuilder.defaultClient();

        try {
            String sessionId = createSession();
            System.out.println("Created a Face Liveness Session with ID: " +
 sessionId);

            String status = getSessionResults(sessionId);
            System.out.println("Status of Face Liveness Session: " + status);

        } catch(AmazonRekognitionException e) {
           e.printStackTrace();
        }
    }

    private static String createSession() throws Exception {

        CreateFaceLivenessSessionRequest request = new
 CreateFaceLivenessSessionRequest();
        CreateFaceLivenessSessionResult result =
 rekognitionClient.createFaceLivenessSession(request);

        String sessionId = result.getSessionId();
        System.out.println("SessionId: " + sessionId);

        return sessionId;
    }
```

```
    private static String getSessionResults(String sessionId) throws Exception {

        GetFaceLivenessSessionResultsRequest request = new
 GetFaceLivenessSessionResultsRequest().withSessionId(sessionId);
        GetFaceLivenessSessionResultsResult result =
 rekognitionClient.getFaceLivenessSessionResults(request);

        Float confidence = result.getConfidence();
        String status = result.getStatus();

        System.out.println("Confidence: " + confidence);
        System.out.println("status: " + status);

        return status;
    }
}
```

Node.Js

下列程式碼片段示範如何使用 AWS Node.Js SDK 呼叫臉部活力 API：

```
const Rekognition = require("aws-sdk/clients/rekognition");

const rekognitionClient = new Rekognition({ region: "us-east-1" });

async function createSession() {
    const response = await rekognitionClient.createFaceLivenessSession().promise();

    const sessionId = response.SessionId;
    console.log("SessionId:", sessionId);

    return sessionId;
}

async function getSessionResults(sessionId) {
    const response = await rekognitionClient
        .getFaceLivenessSessionResults({
            SessionId: sessionId,
        })
        .promise();
```

```
    const confidence = response.Confidence;
    const status = response.Status;
    console.log("Confidence:", confidence);
    console.log("Status:", status);

    return status;
}

async function main() {
    const sessionId = await createSession();
    console.log("Created a Face Liveness Session with ID:", sessionId);

    const status = await getSessionResults(sessionId);
    console.log("Status of Face Liveness Session:", status);
}

main();
```

Node.Js (Javascript SDK v3)

下面的代碼片段演示了如何使用 AWS Node.Js SDK 調用臉部活性 API

```
import { RekognitionClient, CreateFaceLivenessSessionCommand } from "@aws-sdk/
client-rekognition"; // ES Modules
import const { RekognitionClient, CreateFaceLivenessSessionCommand } =
 require("@aws-sdk/client-rekognition"); // CommonJS import
const client = new RekognitionClient(config);
const input = {
  KmsKeyId: "STRING_VALUE",
  Settings: {
    OutputConfig: { // LivenessOutputConfig
      S3Bucket: "STRING_VALUE", // required
      S3KeyPrefix: "STRING_VALUE",
    },
    AuditImagesLimit: Number("int"),
  },
  ClientRequestToken: "STRING_VALUE",
};
const command = new CreateFaceLivenessSessionCommand(input);
const response = await client.send(command);
// { // CreateFaceLivenessSessionResponse
//   SessionId: "STRING_VALUE", // required
```

```
// };
```

設定和自訂應用程式

設定您的應用程式

您的人臉活體應用程式可以在移動裝置或桌面網路瀏覽器上執行。您需要設定人臉活體元件，以與您選擇的解決方案整合。您還必須確保您的應用程式具有使用裝置攝影機的權限。Amplify 活體指南提供了有關如何進行以下操作的詳細說明：

- 安裝及設定 AWS Amplify
- 匯入和彩現元 FaceLivenessDetector 件
- 聆聽回電
- 轉譯 Amplify 範例錯誤訊息

自訂您的應用

您可以使用 AWS Amplify 自訂活體應用程式的某些元件。

如需翻譯的相關資訊，請參閱 Amplify 驗證器文件。

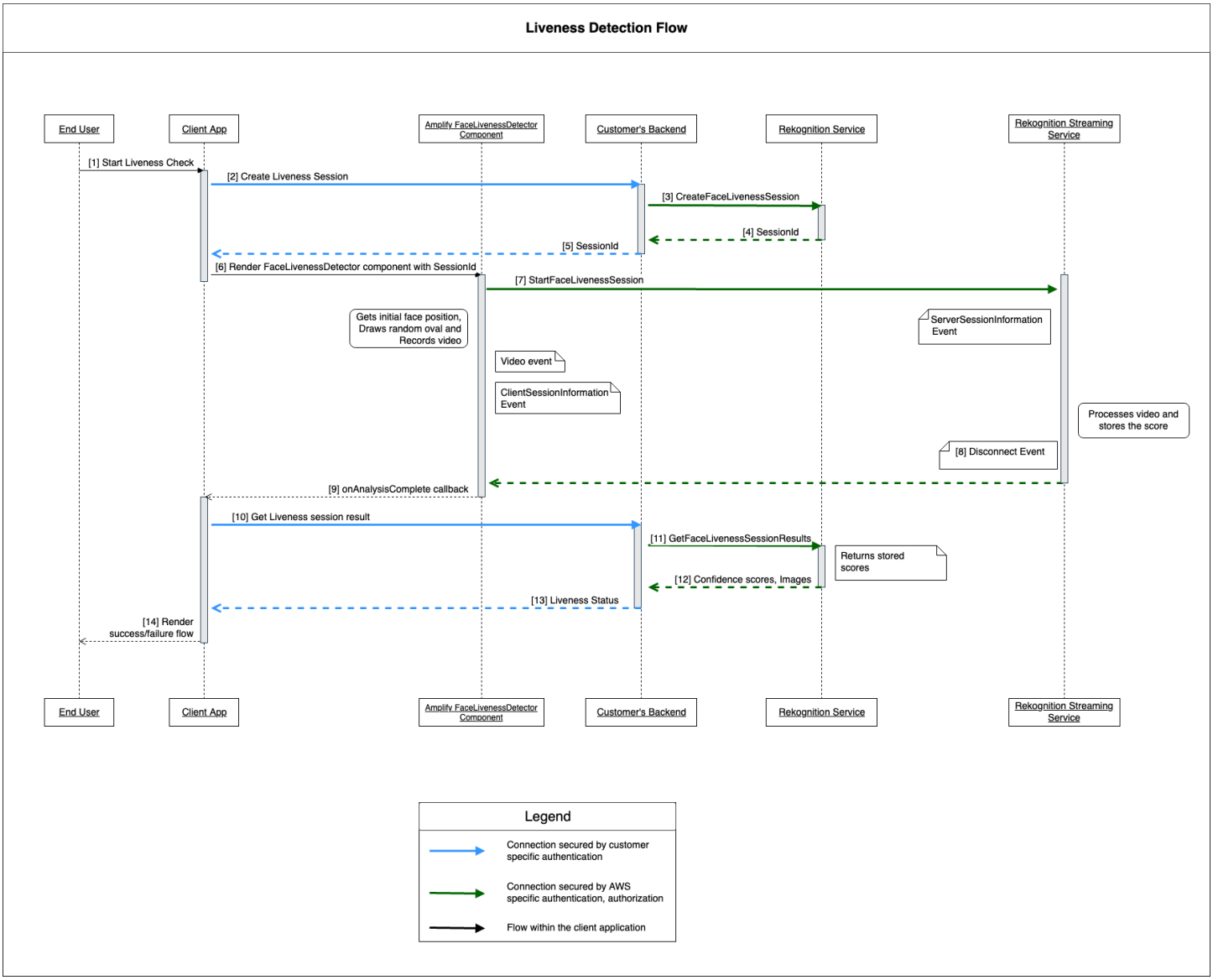
如需有關自訂 Amplify 元件和主題的資訊，請參閱關於主題設定的 Amplify 文件。

人臉活體的共同責任模型

「安全性與法規遵循」是客戶 AWS 與您之間的共同責任。在這裡閱讀有關 AWS 共同責任模式的更多信

1. 對 AWS 服務的所有呼叫（通過客戶端應用程序或客戶後端）都通過 AWS Auth（AWS 身份驗證）進行身份驗證和授權。這是人臉活體服務所有者的責任，以確保這種情況發生。
2. 所有對用戶後端 (從用戶端應用程式) 的呼叫都透過客戶進行身分驗證和授權。由客戶承担该责任。客戶必須確保來自客戶端應用程式的呼叫經過身分驗證，並且沒有以任何方式操縱。
3. 客戶後端必須識別執行人臉活體挑戰的最終使用者。客戶有責任將最終使用者與人臉活體工作階段聯繫起來。人臉活體服務不會區分最終使用者。它只能識別呼叫 AWS 身份（客戶處理的身份）。

下列流程圖顯示 AWS 服務或客戶驗證哪些呼叫：



對 Amazon Rekognition 臉部活力服務的所有呼叫都受到 AWS 身份驗證 (使用 AWS 簽署機制) 的保護。這包含下列呼叫：

- [3] CreateFaceLivenessSessionAPI 呼叫 (來自客戶的後端)
- [7] StartFaceLivenessSessionAPI 呼叫 (來自用戶端應用程式)
- [11] GetFaceLivenessSessionResultsAPI 呼叫 (來自客戶的後端)

所有對客戶後端的呼叫都需要具有身分驗證和授權機制。客戶需要確保使用的第三方程式碼/資料庫/等正在積極維護和開發。客戶還需要確保正確的最終使用者呼叫正確的人臉活體工作階段。客戶必須驗證並授權以下流程：

- [2] 建立人臉活體工作階段 (來自客戶端的應用程式)
- [10] 獲取人臉活體工作階段結果 (來自客戶端的應用程式)

客戶可以遵循 STRIDE 安全模型，以確保其 API 呼叫受到保護。

Type	描述	安全控制
欺騙	針對存取和使用其他使用者認證的威脅動作，例如使用者名稱和密碼。	身分驗證
篡改	意圖惡意變更或修改持續性資料的威脅動作。範例包括資料庫中的記錄，以及透過開放網路 (例如網際網路) 在兩台電腦之間傳輸中的資料變更。	完整性
否认	旨在在缺乏追蹤作業能力的系統中執行禁止作業的威脅動作。	不可否认
資訊揭露	意圖讀取未被授權存取或讀取傳輸中資料的檔案的威脅動作。	保密
拒絕服務	嘗試拒絕存取有效使用者的安全威脅處理行動，例如讓 Web 伺服器暫時無法使用或無法使用。	可用性
權限提升	威脅動作旨在取得資源的特權存取權，以取得資訊的未經授權存取或危及系統。	授權

AWS 通過以下方式保護其連接：

1. 計算請求簽名，然後在服務端驗證簽名。請求使用此簽名進行身分驗證。

2. AWS 客戶必須設置適當的 IAM 角色以授權某些操作/操作。呼叫 AWS 服務時需要這些 IAM 角色。
3. 只允許使用 HTTPS 要求提 AWS 供服務。請求在開放的網路中使用 TLS 進行加密。這樣可以保護請求的機密性並維持其完整性。
4. AWS 服務記錄足夠的數據來識別客戶撥打的呼叫。這樣可以防止否認攻擊。
5. AWS 服務擁有保持足夠可用性

客戶有責任透過下列方式保護其服務和 API 呼叫：

1. 客戶必須確保他們遵循適當的驗證機制。有跡象表明，可用於驗證請求的各種身分驗證機制。客戶可以探索基於身份驗證的摘要、OAuth、OpenID 連接和其他機制。
2. 客戶必須確定其服務支援適當的加密通道 (例如 TLS/HTTPS)，才能進行服務 API 呼叫。
3. 客戶必須確定其記錄了必要的資料，專門用於識別 API 呼叫和呼叫者。他們應該能夠使用定義的參數和調用時間來識別調用其 API 的客戶端。
4. 客戶必須確保其系統可用，能防禦 DDoS 攻擊。以下是一些針對 DDoS 攻擊的防禦技術範例。

客戶有責任保留其應用程式 up-to-date。如需更多詳細資訊，請參閱 人臉活體更新指南。

人臉活體更新指南

AWS 定期更新 Face Liveness AWS SDK（用於客戶後端）和 Ampli AWS fy SDK（用於客戶端應用程序）的 FaceLivenessDetector 組件，以提供新功能，更新的 API，增強的安全性，錯誤修復，可用性改進等。我們建議您保留 SDK， up-to-date 以確保功能的最佳運作。如果您繼續使用舊版本的 SDK，則出於可維護性和安全性原因，可能會阻止請求。

面部活力要求您使用該 FaceLivenessDetector 組件，包括在 Ampli AWS fy SDK（反應，iOS，安卓系統）。

版本控制和時間框架

我們正在版本控制人臉活體特徵的以下關鍵組成部分。我們遵循語義版本控制格式。例如，關於 X、Y、Z 的版本格式，X 表示主要版本，Y 表示次要版本，Z 則表示補丁版本。

- 面對活力用戶挑戰（例如FaceMovementAndLight挑戰挑戰）是 API 的一部分 StartFaceLivenessSession
- FaceLivenessDetector 透過 AWS Amplify SDK 提供的元件可用於用戶端應用程式

主要版本：我們為關鍵安全性、破壞 API 和顯示阻止可用性更新，保留主要版本更新。應用程式和客戶後端必須盡快更新，您才能繼續使用人臉活體特徵。發佈了新的主要版本後，我們將從新版本發布之日起 120 天內支持以前的主要版本。我們可能會在 120 天後阻止來自以前的主要版本的請求。

次要版本：我們為重要的安全性和可用性特徵和改進，保留次要版本更新。我們強烈建議您使用更新版本。儘管我們努力確保次要更新盡可能長時間向後兼容，但我們可能會在新的次要版本發布 180 天後宣布 end-of-support 以前的次要版本。

補丁版本：我們為漏洞修復和改進，保留補丁版本更新。雖然我們建議您保留版本以 up-to-date獲得最佳安全性和使用者體驗，但我們會努力確保修補程式更新完全向後相容，直到我們發佈新的主要或次要版本為止。

版本控制時間窗口 (主要為 120 天，未成年人為 180 天) 適用於更新應用程式中的 SDK，將您的應用程式上傳到應用程式商店或網站以及下載最新版應用程式的使用者。

版本發布和兼容性矩陣

FaceLivenessDetector 組件或用戶挑戰的主要版本的發布通常一致。若要協助您追蹤版本相依性，請參閱下表中連結的資源。

SDK 版本和變更記錄：

FaceLivenessDetector 適用於網頁 SDK		FaceLivenessDetector 適用於 iOS 開發套	FaceLivenessDetector 適用於安卓 SDK
目前版本	更新日誌	目前版本/變更記錄檔	目前版本/變更記錄檔

使用者挑戰：

挑戰名稱	版本	版本日期	退休日期
FaceMovementAndLight挑戰	v1.0.0	4/10/2023	N/A

新版本的通訊

AWS 透過下列管道傳達新版本：

- 服務健康狀態更新電子郵件通知會傳送至與人臉活體帳號 ID 相關聯的帳戶電子郵件。
- 已在個別存 GitHub 放庫發佈 AWS SDK 的更新和相關通知。
- 針對個別 GitHub 存放庫的 AWS Amplify SDK 和相關通知發佈的已發佈更新。

我們建議您訂閱這些頻道留下來 up-to-date。

人臉活體常見問答集

使用下列常見問題集專案，尋找有關 Rekognition 人臉活體的常見問題解答。

- 人臉活體檢查後會輸出什麼？

  Rekognition 人臉活體為每次活性檢查提供以下輸出：

  - 可信度分數：傳回 0 到 100 的數值分數。該分數表示該自拍影片來自真人的可能性，而非存在欺騙的不良人士。
  - 高質量映像：從自拍影片中提取單個高質量映像。此框架可用於各種目的，例如人臉比較，年齡估計或人臉搜尋。
  - 審核映像：自拍影片最多傳回四張映像，可用於稽核記錄目的。
- Rekognition 人臉活體是否符合 iBeta 呈現攻擊偵測 (PAD) 測試？

  iBeta 品質保證的呈現攻擊偵測 (PAD) 測試根據 ISO/IEC 30107-3 進行。iBeta 通過 NIST/NVLAP 認證，可測試並提供符合此 PAD 標準的結果。Rekognition 人臉活體通過第 1 級和 2 級 iBeta 呈現攻擊偵測 (PAD) 的一致性測試，並獲得較高的 PAD 分數。報告可見於 iBeta 網頁的此處。
- 如何獲得高質量的影格和其他影格？

  視 CreateFaceLivenessSessionAPI 請求的組態而定，高品質框架和其他框架可以作為原始位元組傳回或上傳到您指定的 Amazon S3 儲存貯體。
- 我可以更改橢圓形和彩色燈光的位置嗎？

  不可以，橢圓形的位置和彩色燈具有安全特徵，因此無法定制。
- 我可以根據應用程式自訂使用者界面嗎？

可以，您可以自訂大部分的螢幕元件，例如主題、色彩、語言、文字內容和字型，以符合您的應用程式。有關如何自訂這些元件的詳細資訊，可以在我們的 React、Swift 和 Android UI 元件文件中找到。

- 我可以自訂倒計時和將人臉置於橢圓形中的時間嗎？

  不可以，倒計時和人臉匹配的時間是根據 1000 次使用者檢查的大規模內部研究預先確定的，目標是在安全性和延遲之間提供最佳平衡。因此，無法自訂這些時間設定。

- 為什麼人臉橢圓形位置並不總是居中？

  作為安全措施，每次檢查時橢圓形的位置設計會變化。這種動態定位增強了人臉活體的安全性。

- 為什麼在某些情況下，橢圓形會溢出到顯示區域上方？

  每次檢查都會更改橢圓形位置，以提高安全性。有時，橢圓形可能會溢出在顯示區域上。但是，人臉活體元件可確保任何溢出受到限制，並保留使用者完成檢查的能力。

- 不同的顏色燈光是否符合輔助使用指南？

  是的，我們產品中的不同顏色燈符合 WCAG 2.1 中列出的輔助使用指南。經過超過 1000 次使用者檢查的驗證，使用者體驗每秒會顯示大約兩種顏色，符合將色彩限制為每秒三種的建議。這減少了大多數人口觸發癲癇發作的可能性。

- SDK 是否會調整螢幕亮度以獲得最佳效果？

  人臉活體的移動式 SDK (適用於 Android 和 iOS) 會在開始檢查時自動調整亮度。但是，對於 Web SDK，網頁上存在阻止自動亮度調整的限制。在這種情況下，我們希望 Web 應用程式指示最終使用者手動增加熒幕亮度以獲得最佳效果。

- 必須是橢圓形嗎？ 我們可以使用其他類似的形狀嗎？

  不可以，橢圓形的大小、形狀和位置無法自訂。特定的橢圓形設計經過精心挑選，因為其能高效地準確捕捉和分析人臉操作。因此，橢圓形不能修改。

- 什麼是 end-to-end 延遲？

  我們從使用者啟動完成活動檢查所需的動作到使用者取得結果的時間 (通過或失敗)，測量 end-to-end 延遲。在最好的情況下，延遲是 5 秒。在一般情況下，我們預計延遲大約是 7 秒。在最壞的情

況下，延遲是 11 秒。我們看到 end-to-end 延遲的變化，因為它取決於：用戶完成所需操作的時間（即，將他們的臉移到橢圓形），網絡連接，應用程序延遲等。

- 我可以在沒有 Amplify SDK 的情況下使用人臉活體特徵嗎？

  不可以，使用 Rekognition 人臉活體特徵需要 Amplify SDK。

- 我在哪裡可以找到與人臉活體相關的錯誤狀態？

  您可以在此處看到不同的人臉活體錯誤狀態。

- 我所在的區域不提供人臉活體。如何使用此特徵？

  您可以選擇在任何可用的區域呼叫人臉活體，具體取決於您的流量承載和接近程度。臉部活力目前在以下 AWS 地區提供：

  - 美國東部 (維吉尼亞北部)
  - 美國西部 (奧勒岡)
  - 歐洲 (愛爾蘭)
  - 亞太區域 (東京、孟買)

  即使您的 AWS 帳戶位於不同的地區，預計延遲差異也不會顯著。您可以透過 Amazon S3 位置或原始位元組取得高品質的自拍畫面和稽核影像，但 Amazon S3 儲存貯體必須與臉部生命 AWS 區域相符。如果兩者不同，則必須以原始位元組形式接收映像。

- Amazon Rekognition 活體偵測是否會使用客戶內容來改善服務？

  您可以透過 AWS Organizations 退出政策，選擇不使用映像和影片輸入來改善或開發 Rekognition 和其他 Amazon 機器學習/人工智慧技術的品質。如需有關如何選擇退出的詳細資訊，請參閱管理 AI 服務選擇退出政策。

批量分析

Amazon Rekognition 大量分析可讓您在作業中使用資訊清單檔案，以非同步方式處理大量映像集合。StartMediaAnalysisJob每個影像的輸出與您分析操作傳回的輸出相符。

目前，Rekognition 支援使用此作業進行分析。DetectModerationLabels

按作業已處理完成的影像數量向您收取費用。將已完成任務的結果會輸出到指定的 Amazon S3 儲存貯體。

請注意，批量分析不支援 Amazon A2I 整合。

該 API 可以檢測動畫或插圖內容類型，並返回有關檢測到的內容類型的信息作為響應的一部分。

批量處理映像

您可以提交資訊清單檔案並呼叫作業，以啟動新的大量分析工 StartMediaAnalysisJob 作。輸入資訊清單檔案包含 Amazon S3 儲存貯體中映像檔的參考，其格式如下：

```
{"source-ref": "s3://foo/bar/1.jpg"}
```

建立批量分析作業 (CLI)

1. 如果您尚未執行：

   a. 建立或更新具有 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 許可的使用者。如需詳細資訊，請參閱 步驟 1：設定 AWS 帳戶並建立使用者。

   b. 安裝和設定 AWS CLI AWS 軟體開發套件。如需詳細資訊，請參閱 步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 將影像上傳至您的 S3 儲存貯體。

   如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

3. 使用下列指令建立和擷取批量分析工作。

CLI

使用下列指令來呼叫StartMediaAnalysisJob作業以進行分析的 DetectModerationLabels 作業：

```
# Requests
# Starting DetectModerationLabels job with default settings
aws rekognition start-media-analysis-job \
--operations-config "DetectModerationLabels={MinConfidence='1'}" \
--input "S3Object={Bucket=my-bucket,Name=my-input.jsonl}" \
--output-config "S3Bucket=my-output-bucket,S3KeyPrefix=my-results"
```

您可以使用GetMediaAnalysisJob操作取得有關指定任務的資訊，例如存放結果和摘要檔案的儲存貯體的 Amazon S3 路徑。您提供 StartMediaAnalysisJob 或傳回的工作 ID ListMediaAnalysisJob。有關作業的詳細資料僅保留一年。

```
# Request
aws rekognition get-media-analysis-job \
--job-id customer-job-id
```

您可以使用傳回工單頁面的工單ListMediaAnalysisJobs作業，列出所有大量分析。使用 `max-results` 引數時，您可以指定每頁傳回的最大作業數目，數目僅限於 `max-results`。每頁最多傳回 100 個結果。有關作業的詳細資料僅保留一年。

```
# Request
# Specify number of jobs to return per page, limited to max-results.
aws rekognition list-media-analysis-jobs --max-results 1
```

StartMediaAnalysisJob 輸出清單

批量分析作業會產生包含作業結果的輸出資訊清單檔案，以及資訊清單摘要，其中包含處理輸入資訊清單專案時任何錯誤的統計資料和詳細資訊。

如果輸入的清單檔案中包含重複的專案，將不會嘗試透過作業篩選出唯一的輸入，而是會處理所有提供的專案。

輸出資訊清單檔案的格式如下：

```
// Output manifest for content moderation
{"source-ref":"s3://foo/bar/1.jpg", "detect-moderation-labels":
 {"ModerationLabels":[],"ModerationModelVersion":"7.0","ContentTypes":
[{"Confidence":72.7257,"Name":"Animated"}]}}
```

輸出清單檔案摘要的格式如下：

```
{
    "version": "1.0",                # Schema version, 1.0 for GA.
    "statistics": {
        "total-json-lines": Number,  # Total number json lines (images) in the input
 manifest.
        "valid-json-lines": Number,  # Total number of JSON Lines (images) that contain
 references to valid images.
        "invalid-json-lines": Number # Total number of invalid JSON Lines. These lines
 were not handled.
    },
    "errors": [
        {
            "line-numer": Number,   # The number of the line in the manifest where the
 error occured.
            "source-ref": "String", # Optional. Name of the file if was parsed.
            "code": "String",       # Error code.
            "message": "String"     # Description of the error.
        }
     ]
}
```

內容類型

作業會傳回有關 StartMediaAnalysisJob 作業所分析之媒體內容類型的 GetMediaAnalysisJob 資訊。ContentType 可以是兩個不同類別之一：

- 動畫內容，包括電子遊戲和動畫（例如卡通，漫畫，漫畫，動漫）。
- 插圖內容，其中包括繪畫，繪畫和素描。

預測驗證和轉接器訓練

您也可以透過 Rekognition 主控台利用批量分析來取得一批影像的預測、預測驗證，然後使用已驗證的預測建立轉接器。轉接器可讓您增強任何支援的 Rekognition 作業的準確性。

目前，您可以建立轉接器，與 Rekognition 自訂內容管制特徵搭配使用。藉由建立轉接器並將其提供給DetectModerationLabels作業，您可以針對與特定使用案例相關的內容協調工作達到更好的準確性。

如需自訂內容管制的詳細資訊，請參閱 透過自訂管制提升準確性。如需使用批量分析驗證預測的說明，請參閱 批量分析和驗證。如需有關如何使用 Rekognition 主控台驗證預測和建立轉接器的教學課程，請參閱 自訂管制轉接器教學課程。

教學課程

這些跨服務教學課程示範如何將 Rekognition 的 API 作業與其他作業搭配使用AWS用於建立範例應用程式並完成各種工作的服務。這些教程中的大多數都使用 Amazon S3 來存放圖像或視頻。其他常用服務包括AWS Lambda。

主題

使用亞馬遜 RDS 和動態 B 存放亞馬遜重新認知資料

使用 Amazon Rekognition 的 API 時，請務必記住，API 操作不會儲存任何產生的標籤。您可以通過將這些標籤與相應圖像的標識符一起放置在數據庫中來保存這些標籤。

本教學課程示範偵測標籤，並將這些偵測到的標籤儲存至資料庫。本教程中開發的示例應用程序將從亞馬遜 S3桶調用DetectLabels對這些圖像進行操作，並將生成的標籤存儲在數據庫中。應用程式會根據您要使用的資料庫類型，將資料儲存在 Amazon RDS 資料庫執行個體或 DynamoDB 資料庫中。

您將使用AWS適用於蟒蛇的 SDK或本教程。您還可以看到AWS說明文件 SDK 範例GitHub回購有關更多 Python 教程。

主題

先決條件

在開始本教學課程之前，您將需要安裝 Python 並完成所需的步驟設置蟒蛇AWSSDK。除此之外，請確保您擁有：

建立 AWS 帳戶和身分與存取權管理角色

安裝了 Python 軟件開發套件

正確配置AWS存取認證

創建的亞馬遜 S3 存儲桶充滿了圖像

建立 RDS 資料庫執行個體，如果使用 RDS 儲存資料

在亞馬遜 S3 存儲桶中獲取圖像的標籤

首先編寫一個函數，該函數將在 Amazon S3 存儲桶中獲取映像的名稱，然後檢索該映像。將顯示此圖像以確認正確的圖像正在傳遞到呼叫DetectLabels這也在功能中。

1. 找到您想要使用的 Amazon S3 儲存貯體，並寫下其名稱。您將撥打這個 Amazon S3 儲存貯體的呼叫，並讀取其中的映像檔。確保您的存儲桶包含一些圖像以傳遞給DetectLabels操作。
2. 撰寫程式碼以連線到您的 Amazon S3 儲存貯體。您可以使用 Boto3 連接到亞馬遜 S3 資源，以從亞馬遜 S3 儲存貯體擷取映像。連線到 Amazon S3 資源後，您可以透過提供儲存貯體方法與 Amazon S3 儲存貯體的名稱來存取儲存貯體。連線到 Amazon S3 儲存貯體之後，您可以使用 Object 方法從儲存貯體擷取映像。通過使用 Matplotlib，您可以使用此連接在處理圖像時可視化圖像。Boto3 也可用來連接至重新認知用戶端。

   在下列程式碼中，將您的地區提供給 region_name 參數。您將傳遞亞馬遜 S3 存儲桶名稱和圖像名稱DetectLabels，它會傳回對應影像的標籤。僅從響應中選擇標籤後，將返回圖像的名稱和標籤。

```
import boto3
from io import BytesIO
from matplotlib import pyplot as plt
from matplotlib import image as mp_img

boto3 = boto3.Session()

def read_image_from_s3(bucket_name, image_name):
```

```
    # Connect to the S3 resource with Boto 3
    # get bucket and find object matching image name
    s3 = boto3.resource('s3')
    bucket = s3.Bucket(name=bucket_name)
    Object = bucket.Object(image_name)

    # Downloading the image for display purposes, not necessary for detection of
 labels
    # You can comment this code out if you don't want to visualize the images
    file_name = Object.key
    file_stream = BytesIO()
    Object.download_fileobj(file_stream)
    img = mp_img.imread(file_stream, format="jpeg")
    plt.imshow(img)
    plt.show()

    # get the labels for the image by calling DetectLabels from Rekognition
    client = boto3.client('rekognition', region_name="region-name")
    response = client.detect_labels(Image={'S3Object': {'Bucket': bucket_name,
 'Name': image_name}},
                                    MaxLabels=10)

    print('Detected labels for ' + image_name)

    full_labels = response['Labels']

    return file_name, full_labels
```

3. 將此代碼保存在一個名為 get_images.py 的文件中。

創建一個亞馬遜動態 B 表

下列程式碼會使用 Boto3 連線到 DynamoDB，並使用 DynamoDB`CreateTable`方法來創建一個名為圖像的表。此資料表有一個複合主索引鍵，其中包含一個名為 Image 的分割索引鍵和一個稱為 Label 的排序索引鍵 「影像」鍵包含影像的名稱，而「標籤」鍵則儲存指定給該影像的標籤。

```
import boto3

def create_new_table(dynamodb=None):
    dynamodb = boto3.resource(
        'dynamodb',)
```

```
    # Table defination
    table = dynamodb.create_table(
        TableName='Images',
        KeySchema=[
            {
                'AttributeName': 'Image',
                'KeyType': 'HASH'  # Partition key
            },
            {
                'AttributeName': 'Labels',
                'KeyType': 'RANGE'  # Sort key
            }
        ],
        AttributeDefinitions=[
            {
                'AttributeName': 'Image',
                'AttributeType': 'S'
            },
            {
                'AttributeName': 'Labels',
                'AttributeType': 'S'
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        }
    )
    return table

if __name__ == '__main__':
    device_table = create_new_table()
    print("Status:", device_table.table_status)
```

將此程式碼儲存在編輯器中，然後執行一次以建立 DynamoDB 表格。

將資料上傳至

現在已建立 DynamoDB 資料庫，而且您擁有取得映像標籤的函數，您可以將標籤存放在 DynamoDB 中。下列程式碼會擷取 S3 儲存貯體中的所有映像、取得它們的標籤，然後將資料存放在 DynamoDB 中。

1. 您需要撰寫程式碼，以便將資料上傳至 DynamoDB。一個叫做的函數get_image_names用於連接到 Amazon S3 存儲桶，並將存儲桶中所有映像的名稱作為列表返回。您會將此清單傳遞至read_image_from_S3函數，這是從get_images.py您建立的檔案。

```
import boto3
import json
from get_images import read_image_from_s3

boto3 = boto3.Session()

def get_image_names(name_of_bucket):

    s3_resource = boto3.resource('s3')
    my_bucket = s3_resource.Bucket(name_of_bucket)
    file_list = []
    for file in my_bucket.objects.all():
        file_list.append(file.key)
    return file_list
```

2. 該read_image_from_S3我們之前創建的函數將返回正在處理的圖像的名稱以及與該圖像相關的標籤的字典。一個叫做的函數find_values用於僅從響應中獲取標籤。然後，映像的名稱及其標籤就可以上傳到 DynamoDB 表。

```
def find_values(id, json_repr):
    results = []

    def _decode_dict(a_dict):
        try:
            results.append(a_dict[id])
        except KeyError:
            pass
        return a_dict

    json.loads(json_repr, object_hook=_decode_dict) # Return value ignored.
    return results
```

3. 您將使用第三個函數，稱為load_data，將影像和標籤實際載入您建立的 DynamoDB 表格中。

```
def load_data(image_labels, dynamodb=None):

    if not dynamodb:
        dynamodb = boto3.resource('dynamodb')
```

```
    table = dynamodb.Table('Images')

    print("Adding image details:", image_labels)
    table.put_item(Item=image_labels)
    print("Success!!")
```

4. 這是我們之前定義的三個函數被調用的地方，並執行操作。將上面定義的三個函數與下面的代碼一起添加到 Python 文件中。執行程式碼。

```
bucket = "bucket_name"
file_list = get_image_names(bucket)

for file in file_list:
    file_name = file
    print("Getting labels for " + file_name)
    image_name, image_labels = read_image_from_s3(bucket, file_name)
    image_json_string = json.dumps(image_labels, indent=4)
    labels=set(find_values("Name", image_json_string))
    print("Labels found: " + str(labels))
    labels_dict = {}
    print("Saving label data to database")
    labels_dict["Image"] = str(image_name)
    labels_dict["Labels"] = str(labels)
    print(labels_dict)
    load_data(labels_dict)
    print("Success!")
```

您剛剛使用過DetectLabels，為您的影像產生標籤，並將這些標籤儲存在 DynamoDB 執行個體中。確保您拆除了在閱讀本教程時創建的所有資源。這樣可以防止您為未使用的資源收取費用。

在亞馬遜 RDS 中創建一個 MySQL 數據庫

在進一步之前，請確保您已完成安裝程序對於亞馬遜 RDS 和創建了一個 MySQL 數據庫實例使用亞馬遜 RDS。

下面的代碼利用PyMySQL程式庫和您的 Amazon RDS 資料庫執行個體。它創建一個表格來保存圖像的名稱以及與這些圖像相關聯的標籤。Amazon RDS 會接收用於建立資料表和將資料插入資料表的命令。若要使用 Amazon RDS，您必須使用主機名稱、使用者名稱和密碼連接到 Amazon RDS 主機。您將通過提供這些參數連接到亞馬遜 RDSPyMySQL 的`connect`函數和創建一個光標的實例。

1. 在下列程式碼中，將主機的值取代為 Amazon RDS 主機端點，並以與 Amazon RDS 執行個體關聯的主使用者名稱取代使用者的值。您還需要將密碼替換為主要用戶的主密碼。

```
import pymysql

host = "host-endpoint"
user = "username"
password = "master-password"
```

2. 建立要插入影像和標籤資料的資料庫和表格。透過執行並提交建立查詢來執行此操作。下面的代碼創建一個數據庫。只執行一次此程式碼。

```
conn = pymysql.connect(host=host, user=user, passwd=password)
print(conn)
cursor = conn.cursor()
print("Connection successful")

# run once
create_query = "create database rekogDB1"
print("Creation successful!")
cursor.execute(create_query)
cursor.connection.commit()
```

3. 建立資料庫之後，您必須建立一個表格來插入影像名稱和標籤。要創建一個表，您首先將 use SQL 命令與數據庫的名稱一起傳遞給`execute`功能。建立連線之後，會執行建立資料表的查詢。下面的代碼連接到數據庫，然後創建一個包含兩個主鍵的表，稱為`image_id`，以及儲存標籤的文字屬性。使用您之前定義的導入和變量，並運行此代碼在數據庫中創建一個表。

```
# connect to existing DB
cursor.execute("use rekogDB1")
cursor.execute("CREATE TABLE IF NOT EXISTS test_table(image_id VARCHAR (255)
 PRIMARY KEY, image_labels TEXT)")
conn.commit()
print("Table creation - Successful creation!")
```

將數據上傳到亞馬遜 RDS MySQL 表

在資料庫中建立 Amazon RDS 資料庫和表格之後，您可以取得映像的標籤，並將這些標籤存放在 Amazon RDS 資料庫中。

1. 連接到 Amazon S3 儲存貯體，並擷取儲存貯體中所有映像的名稱。這些圖像名稱將被傳遞到`read_image_from_s3`您之前創建的功能以獲取所有圖像的標籤。下列程式碼會連線至 Amazon S3 儲存貯體，並傳回儲存貯體中所有映像的清單。

```
import pymysql
from get_images import read_image_from_s3
import json
import boto3

host = "host-endpoint"
user = "username"
password = "master-password"

conn = pymysql.connect(host=host, user=user, passwd=password)
print(conn)
cursor = conn.cursor()
print("Connection successful")

def get_image_names(name_of_bucket):

    s3_resource = boto3.resource('s3')
    my_bucket = s3_resource.Bucket(name_of_bucket)
    file_list = []
    for file in my_bucket.objects.all():
        file_list.append(file.key)
    return file_list
```

2. 來自的響應DetectLabelsAPI 不僅包含標籤，因此請編寫一個函數來僅提取標籤值。下面的函數返回一個只包含標籤的列表。

```
def find_values(id, json_repr):
    results = []

    def _decode_dict(a_dict):
        try:
            results.append(a_dict[id])
        except KeyError:
            pass
        return a_dict

    json.loads(json_repr, object_hook=_decode_dict) # Return value ignored.
    return results
```

3. 您將需要一個函數來插入圖像名稱和標籤到您的表中。下面的函數運行插入查詢，並插入任何給定的一對圖像名稱和標籤。

```
def upload_data(image_id, image_labels):

    # insert into db
    cursor.execute("use rekogDB1")
    query = "INSERT IGNORE INTO test_table(image_id, image_labels) VALUES (%s, %s)"
    values = (image_id, image_labels)
    cursor.execute(query, values)
    conn.commit()
    print("Insert successful!")
```

4. 最後，您必須運行上面定義的函數。在以下代碼中，將收集存儲桶中所有圖像的名稱並提供給調用的函數DetectLabels。之後，標籤和它們套用的映像名稱會上傳到您的 Amazon RDS 資料庫。將上面定義的三個函數與下面的代碼一起複製到 Python 文件中。運行蟒蛇文件。

```
bucket = "bucket-name"
file_list = get_image_names(bucket)

for file in file_list:
    file_name = file
    print("Getting labels for " + file_name)
    image_name, image_labels = read_image_from_s3(bucket, file_name)
    image_json = json.dumps(image_labels, indent=4)
    labels=set(find_values("Name", image_json))
    print("Labels found: " + str(labels))
    unique_labels=set(find_values("Name", image_json))
    print(unique_labels)
    image_name_string = str(image_name)
    labels_string = str(unique_labels)
    upload_data(image_name_string, labels_string)
    print("Success!")
```

您已成功使用DetectLabels為您的圖像生成標籤，並使用亞馬遜 RDS 將這些標籤存儲在 MySQL 數據庫中。確保您拆除了在閱讀本教程時創建的所有資源。這樣可以防止您針對未使用的資源收費。

欲了解更多AWS多服務範例，請參閱AWS說明文件 SDK 範例GitHub儲存庫。

使用 Amazon Rekognition 和 Lambda 在 Amazon S3 儲存貯體中標記資產

在本教學中，您會建立自動標記位於 Amazon S3 儲存貯體中的數位資產的 AWS Lambda 函數。Lambda 函數可讀取指定 Amazon S3 儲存貯體中的所有物件。對於儲存貯體中的每個物件，它會將映像傳遞至 Amazon Rekognition 服務，以建立一系列標籤。每個標籤都用來建立套用至影像的標籤。執行 Lambda 函數後，其會根據指定 Amazon S3 儲存貯體中的所有映像自動建立標籤，並將其套用至映像檔。

例如，假設您執行 Lambda 函數，而 Amazon S3 儲存貯體中有這個映像。

然後，應用程式會自動建立標籤並將其套用至影像。

Tags (6)

Track storage cost of other criteria by tagging your objects. **Learn more**

Key	Value
Nature	99.99188
Volcano	97.60948
Eruption	96.54574
Lava	79.63064
Mountain	99.99188
Outdoors	99.99188

> **Note**
>
> 您在本教學課程中使用的服務屬於 AWS 免費方案的一部分。完成教學課程後，我們建議您終止在教學課程中建立的任何資源，這樣就不會被收取任何費用。

本教學課程使用適用於 Java 版本 2 的 AWS SDK。如需其他 Java V2 教學課程，請參閱AWS 文件 SDK 範例 GitHub 儲存庫。

主題

必要條件

在開始之前，您需要完成設定 Java AWS SDK 中的步驟。請確定執行下列操作：

- Java 1.8 JDK.
- Maven 3.6 或更高版本。
- 一個 Amazon S3 儲存貯體，其中包含 5 至 7 個自然映像。這些映像由 Lambda 函數讀取。

設定 IAM Lambda 角色

本教學使用 Amazon Rekognition 和 Amazon S3 服務。將 Lambda 支援 角色設定為具有可讓其從 Lambda 函數調用這些服務的政策。

設定角色

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 在導覽窗格中，選擇角色，然後選擇建立角色。
3. 選擇AWS 服務，接著選擇Lambda。
4. 選擇許可標籤。
5. 搜尋 AWSLambdaBasicExecutionRole。
6. 選擇下一個標籤。

7. 選擇檢閱。
8. 對角色 Lambda 支援命名。
9. 選擇建立角色。
10. 選擇 Lambda 支援來檢視概觀頁面。
11. 選擇連接政策。
12. AmazonRekognitionFullAccess從策略清單中選擇。
13. 選擇連接政策。
14. 搜尋亞馬遜 S3 FullAccess，然後選擇 [附加原則]。

建立專案

建立一個新的 Java 專案，然後使用所需的設定和相依性設定 Maven pom.xml。請確定您的 pom.xml 檔案如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.example</groupId>
<artifactId>WorkflowTagAssets</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<name>java-basic-function</name>
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>bom</artifactId>
            <version>2.10.54</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
```

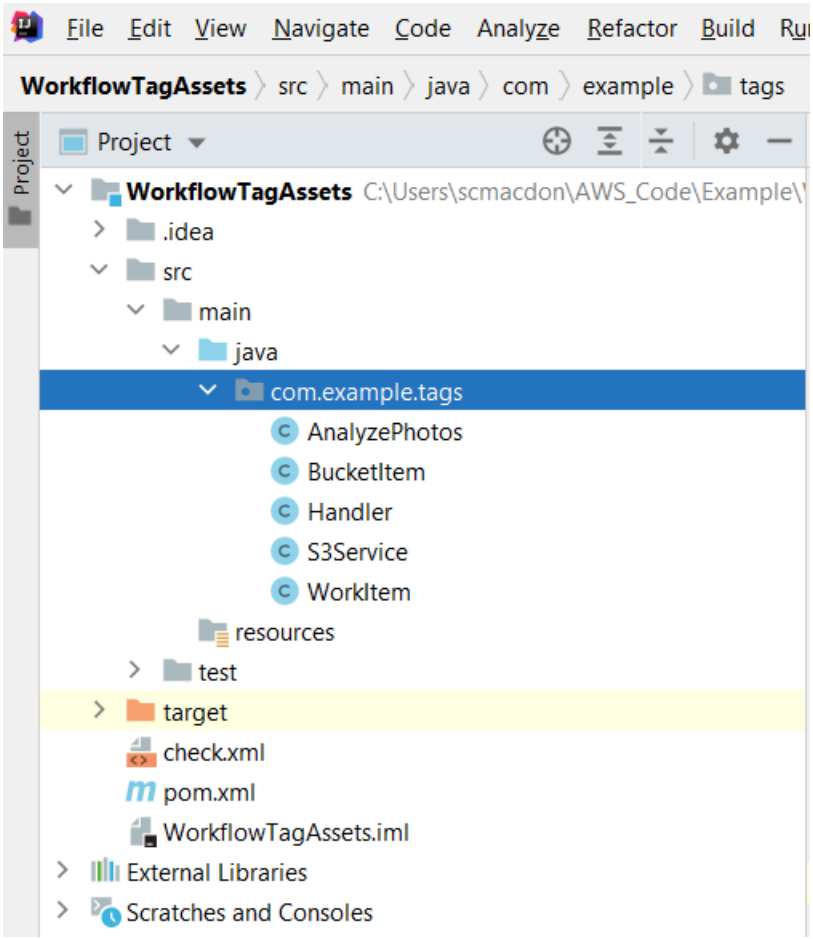
```
    </dependencies>
</dependencyManagement>
<dependencies>
   <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-lambda-java-core</artifactId>
        <version>1.2.1</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.10.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.13.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j18-impl</artifactId>
        <version>2.13.3</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.6.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.6.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.googlecode.json-simple</groupId>
```

```
        <artifactId>json-simple</artifactId>
        <version>1.1.1</version>
    </dependency>
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>s3</artifactId>
    </dependency>
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>rekognition</artifactId>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.2</version>
            <configuration>
                <createDependencyReducedPom>false</createDependencyReducedPom>
            </configuration>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>shade</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
```

```
  </build>
</project>
```

撰寫程式碼

使用 AWS Lambda 執行階段 Java API 來建立定義 Lambda 函數的 Java 類別。在此範例中，有一個名為處理常式的 Lambda 函數的 Java 類別，以及此使用案例所需的其他類別。下圖顯示專案中的 Java 類別。請注意，所有的 Java 類別都位於一個名為com.example.tags的套件中。



為程式碼建立下列 Java 類別：

- 處理常式會使用 Lambda Java 執行階段 API，並執行 AWS 本教學課程中所述的使用案例。執行的應用程式邏輯位於 handleRequest 方法中。
- S3Service 使用 Amazon S3 API 來執行 S3 操作。
- AnalyzePhotos使用 Amazon Rekognition API 來分析圖像。
- BucketItem定義存放 Amazon S3 儲存貯體資訊的模型。
- WorkItem定義存放亞馬遜重新認知資料的模型。

處理常式類別：

這個 Java 程式碼代表處理常式類別。此類別會讀取傳遞至 Lambda 函數的旗標。該 S3 服務。ListBucketObjects方法返回一個 List 對象，其中每個元素都是代表對象鍵的字符串值。如果旗標值為真，則會透過反覆執行清單並呼叫 s3Service.tagAssets 方法，將標籤套用至每個物件，以套用標籤。如果標誌值為假，則 S3 服務。 deleteTagFrom對象方法被調用刪除的標籤。此外，請注意，您可以使用LambdaLogger物件將訊息 CloudWatch 記錄到 Amazon 日誌。

> **Note**
>
> 請務必將儲存貯體名稱指派給 bucketName 變數。

```
package com.example.tags;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class Handler implements RequestHandler<Map<String,String>, String> {

@Override
public String handleRequest(Map<String, String> event, Context context) {
    LambdaLogger logger = context.getLogger();
    String delFlag = event.get("flag");
    logger.log("FLAG IS: " + delFlag);
    S3Service s3Service = new S3Service();
    AnalyzePhotos photos = new AnalyzePhotos();

    String bucketName = "<Enter your bucket name>";
    List<String> myKeys = s3Service.listBucketObjects(bucketName);
    if (delFlag.compareTo("true") == 0) {

        // Create a List to store the data.
        List<ArrayList<WorkItem>> myList = new ArrayList<>();

        // loop through each element in the List and tag the assets.
        for (String key : myKeys) {
```

```
            byte[] keyData = s3Service.getObjectBytes(bucketName, key);

            // Analyze the photo and return a list where each element is a WorkItem.
            ArrayList<WorkItem> item = photos.detectLabels(keyData, key);
            myList.add(item);
        }

        s3Service.tagAssets(myList, bucketName);
        logger.log("All Assets in the bucket are tagged!");

    } else {

        // Delete all object tags.
        for (String key : myKeys) {
            s3Service.deleteTagFromObject(bucketName, key);
            logger.log("All Assets in the bucket are deleted!");
        }
     }
    return delFlag;
 }
}
```

S3Service 類

下列類別使用 Amazon S3 API 來執行 S3 操作。例如，此方getObjectBytes法會傳回代表影像的位元組陣列。同樣地，此方listBucketObjects法會傳回 List 物件，其中每個項目都是指定索引鍵名稱的字串值。

```
package com.example.tags;

import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.PutObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.ListObjectsResponse;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectTaggingResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsRequest;
import java.util.ArrayList;
import java.util.List;
```

```
import software.amazon.awssdk.services.s3.model.Tagging;
import software.amazon.awssdk.services.s3.model.Tag;
import software.amazon.awssdk.services.s3.model.GetObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectTaggingRequest;

public class S3Service {

private S3Client getClient() {

   Region region = Region.US_WEST_2;
   return S3Client.builder()
           .region(region)
           .build();
}

public byte[] getObjectBytes(String bucketName, String keyName) {

   S3Client s3 = getClient();

   try {

       GetObjectRequest objectRequest = GetObjectRequest
               .builder()
               .key(keyName)
               .bucket(bucketName)
               .build();

       // Return the byte[] from this object.
       ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
       return objectBytes.asByteArray();

   } catch (S3Exception e) {
       System.err.println(e.awsErrorDetails().errorMessage());
       System.exit(1);
   }
   return null;
}

// Returns the names of all images in the given bucket.
public List<String> listBucketObjects(String bucketName) {

   S3Client s3 = getClient();
   String keyName;
```

```
    List<String> keys = new ArrayList<>();

    try {
        ListObjectsRequest listObjects = ListObjectsRequest
                .builder()
                .bucket(bucketName)
                .build();

        ListObjectsResponse res = s3.listObjects(listObjects);
        List<S3Object> objects = res.contents();

        for (S3Object myValue: objects) {
            keyName = myValue.key();
            keys.add(keyName);
        }
        return keys;

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

// Tag assets with labels in the given list.
public void tagAssets(List myList, String bucketName) {

    try {

        S3Client s3 = getClient();
        int len = myList.size();

        String assetName = "";
        String labelName = "";
        String labelValue = "";

        // Tag all the assets in the list.
        for (Object o : myList) {

            // Need to get the WorkItem from each list.
            List innerList = (List) o;
            for (Object value : innerList) {
```

```
                WorkItem workItem = (WorkItem) value;
                assetName = workItem.getKey();
                labelName = workItem.getName();
                labelValue = workItem.getConfidence();
                tagExistingObject(s3, bucketName, assetName, labelName, labelValue);
            }
        }

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

// This method tags an existing object.
private void tagExistingObject(S3Client s3, String bucketName, String key, String
label, String LabelValue) {

    try {

        // First need to get existing tag set; otherwise the existing tags are
overwritten.
        GetObjectTaggingRequest getObjectTaggingRequest =
GetObjectTaggingRequest.builder()
                .bucket(bucketName)
                .key(key)
                .build();

        GetObjectTaggingResponse response =
s3.getObjectTagging(getObjectTaggingRequest);

        // Get the existing immutable list - cannot modify this list.
        List<Tag> existingList = response.tagSet();
        ArrayList<Tag> newTagList = new ArrayList(new ArrayList<>(existingList));

        // Create a new tag.
        Tag myTag = Tag.builder()
                .key(label)
                .value(LabelValue)
                .build();

        // push new tag to list.
        newTagList.add(myTag);
        Tagging tagging = Tagging.builder()
```

```
                .tagSet(newTagList)
                .build();

        PutObjectTaggingRequest taggingRequest = PutObjectTaggingRequest.builder()
                .key(key)
                .bucket(bucketName)
                .tagging(tagging)
                .build();

        s3.putObjectTagging(taggingRequest);
        System.out.println(key + " was tagged with " + label);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
  }

 // Delete tags from the given object.
 public void deleteTagFromObject(String bucketName, String key) {

    try {

        DeleteObjectTaggingRequest deleteObjectTaggingRequest =
 DeleteObjectTaggingRequest.builder()
                .key(key)
                .bucket(bucketName)
                .build();

        S3Client s3 = getClient();
        s3.deleteObjectTagging(deleteObjectTaggingRequest);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
  }
}
```

AnalyzePhotos 類

下面的 Java 代碼代表的AnalyzePhotos類。這個類別使用 Amazon Rekognition API 來分析映像。

```
package com.example.tags;

import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.ArrayList;
import java.util.List;

public class AnalyzePhotos {

 // Returns a list of WorkItem objects that contains labels.
 public ArrayList<WorkItem> detectLabels(byte[] bytes, String key) {

    Region region = Region.US_EAST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .region(region)
            .build();

    try {

        SdkBytes sourceBytes = SdkBytes.fromByteArray(bytes);

        // Create an Image object for the source image.
        Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

        DetectLabelsRequest detectLabelsRequest = DetectLabelsRequest.builder()
                .image(souImage)
                .maxLabels(10)
                .build();

        DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);

        // Write the results to a WorkItem instance.
```

```
        List<Label> labels = labelsResponse.labels();
        ArrayList<WorkItem> list = new ArrayList<>();
        WorkItem item ;
        for (Label label: labels) {
            item = new WorkItem();
            item.setKey(key); // identifies the photo.
            item.setConfidence(label.confidence().toString());
            item.setName(label.name());
            list.add(item);
        }
        return list;

    } catch (RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
    return null ;
  }
}
```

BucketItem 類

下列 Java 程式碼代表存放 Amazon S3 物件資料的BucketItem類別。

```
package com.example.tags;

public class BucketItem {

 private String key;
 private String owner;
 private String date ;
 private String size ;


 public void setSize(String size) {
    this.size = size ;
 }

 public String getSize() {
    return this.size ;
 }

 public void setDate(String date) {
```

```
    this.date = date ;
 }

 public String getDate() {
    return this.date ;
 }

 public void setOwner(String owner) {
    this.owner = owner ;
 }

 public String getOwner() {
    return this.owner ;
 }

 public void setKey(String key) {
    this.key = key ;
 }

 public String getKey() {
    return this.key ;
 }
}
```

WorkItem 類

下面的 Java 代碼代表的WorkItem類。

```
 package com.example.tags;

 public class WorkItem {

 private String key;
 private String name;
 private String confidence ;

public void setKey (String key) {
    this.key = key;
}

public String getKey() {
    return this.key;
}
```

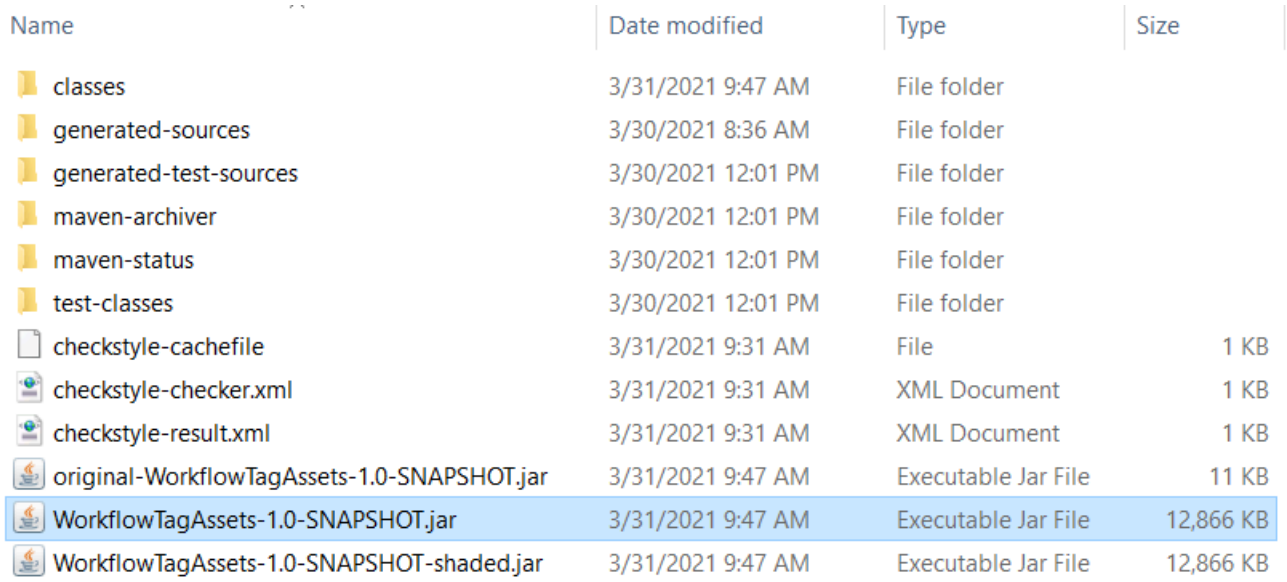
```
public void setName (String name) {
    this.name = name;
}

public String getName() {
    return this.name;
}

public void setConfidence (String confidence) {
    this.confidence = confidence;
}

public String getConfidence() {
    return this.confidence;
}

}
```

Package 專案

透過使用下面的 Maven 命令將該專案打包到一個 .jar (JAR) 檔案。

```
mvn package
```

JAR 檔案位於目標資料夾 (專案資料夾的子資料夾) 中。



Name	Date modified	Type	Size
classes	3/31/2021 9:47 AM	File folder	
generated-sources	3/30/2021 8:36 AM	File folder	
generated-test-sources	3/30/2021 12:01 PM	File folder	
maven-archiver	3/30/2021 12:01 PM	File folder	
maven-status	3/30/2021 12:01 PM	File folder	
test-classes	3/30/2021 12:01 PM	File folder	
checkstyle-cachefile	3/31/2021 9:31 AM	File	1 KB
checkstyle-checker.xml	3/31/2021 9:31 AM	XML Document	1 KB
checkstyle-result.xml	3/31/2021 9:31 AM	XML Document	1 KB
original-WorkflowTagAssets-1.0-SNAPSHOT.jar	3/31/2021 9:47 AM	Executable Jar File	11 KB
WorkflowTagAssets-1.0-SNAPSHOT.jar	3/31/2021 9:47 AM	Executable Jar File	12,866 KB
WorkflowTagAssets-1.0-SNAPSHOT-shaded.jar	3/31/2021 9:47 AM	Executable Jar File	12,866 KB

> **Note**
>
> 請注意專案maven-shade-plugin的 POM 檔案中的使用。此外掛程式負責建立包含所需相依性的 JAR。如果您嘗試在沒有此外掛程式的情況下封裝專案，所需的相依性不會包含在 JAR 檔案中，您將會遇到. ClassNotFoundException

部署 Lambda 函數。

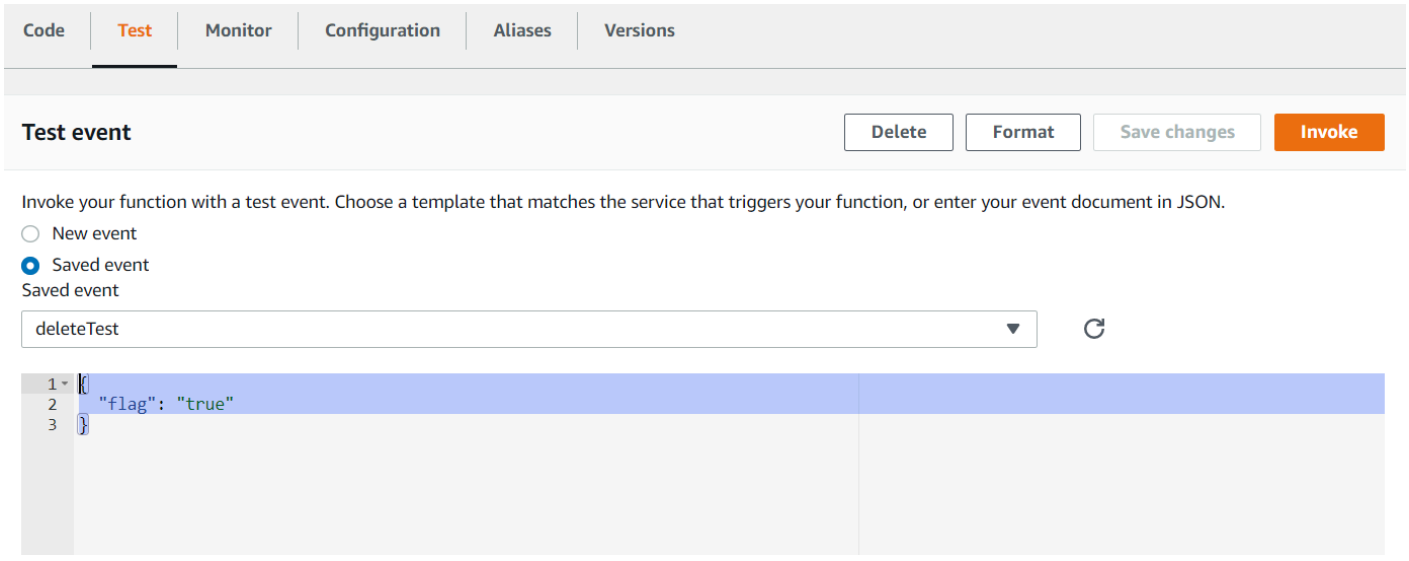
1. 開啟 Lambda 主控台。
2. 選擇 Create Function (建立函數)。
3. 選擇從頭開始撰寫。
4. 在基本資訊區段中，輸入 Cron 做為名稱。
5. 在執行期中，選擇 Java 8。
6. 選擇使用現有角色，然後選擇 Lambda支援 (建立的 IAM 角色)。
7. 選擇建立函數。
8. 針對程式碼專案類型，選擇上傳 .zip 檔案。
9. 選擇上傳，然後瀏覽至您建立的 JAR 檔案。
10. 針對處理程式，請輸入該函數的完整名稱，例如：com.example.tags.Handler:handleRequest (com.example.tags會指定套件，處理程式是後面接著::和方法名稱的類別)。
11. 選擇儲存。

測試 Lambda 方法

在教學課程中，您可以測試 Lambda 函數。

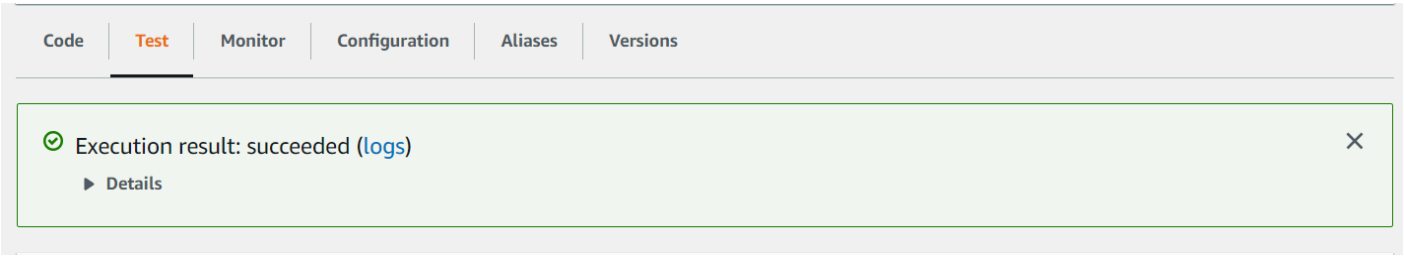
1. 在 Lambda 主控台中，點擊測試索引標籤，然後輸入下列 JSON。

```
                {
"flag": "true"
 }
```



> **Note**
>
> 傳遞真索引標籤的數字資產和傳遞假刪除標籤。

2. 選擇叫用按鈕。調用 Lambda 函數之後，您會看到成功的訊息。



恭喜您，您已建立一個 AWS Lambda 函數，可自動將標籤套用至位於 Amazon S3 儲存貯體中的獨立資產。如同本教學課程開頭所述，請務必終止您在進行本教學課程時所建立的資源，以確保不會產生費用。

如需更 AWS 多多服務範例，請參閱AWS 文件 SDK 範例 GitHub 儲存庫。

建立 AWS 視訊分析儀應用

您可以建立 Java Web 應用程式，透過使用適用於 Java 第 2 版的 AWS SDK 來分析視訊以進行標籤偵測。本教學中建立的應用 AWS 程式可讓您將視訊 (MP4 檔案) 上傳到 Amazon S3 儲存貯體。然後，該應用程式使用 Amazon Rekognition 服務來分析影片。結果會用來填入資料模型，然後使用 Amazon Simple Email Service 產生報告，並透過電子郵件傳送給特定使用者。

下圖顯示應用程式完成影片分析後產生的報告。下表中的欄顯示「年齡範圍」、「鬍鬚」、「眼鏡」和「睜眼」，以及不同屬性預測的信心值。

Age Range	Beard	Eye glasses	Eyes open
AgeRange(Low=38, High=56)	Beard(Value=false, Confidence=83.07253)	Eyeglasses(Value=true, Confidence=55.965977)	EyeOpen(Value=true, Confidence=94.691696)
AgeRange(Low=36, High=52)	Beard(Value=true, Confidence=50.721912)	Eyeglasses(Value=false, Confidence=63.886036)	EyeOpen(Value=true, Confidence=95.906364)
AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=58.38352)	Eyeglasses(Value=false, Confidence=96.39576)	EyeOpen(Value=true, Confidence=53.580643)
AgeRange(Low=49, High=67)	Beard(Value=false, Confidence=81.41662)	Eyeglasses(Value=true, Confidence=65.28722)	EyeOpen(Value=true, Confidence=95.11523)
AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=61.533833)	Eyeglasses(Value=false, Confidence=97.51163)	EyeOpen(Value=true, Confidence=82.21834)
AgeRange(Low=29, High=45)	Beard(Value=false, Confidence=74.22591)	Eyeglasses(Value=true, Confidence=64.906685)	EyeOpen(Value=true, Confidence=98.48175)
AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=65.9394)	Eyeglasses(Value=false, Confidence=94.14824)	EyeOpen(Value=true, Confidence=94.857346)
AgeRange(Low=44, High=62)	Beard(Value=true, Confidence=78.648)	Eyeglasses(Value=true, Confidence=65.83134)	EyeOpen(Value=true, Confidence=98.538666)

在本教學課程中，您會建立叫用各種 AWS 服務的 Spring Boot 應用程式。Spring Boot API 用於構建模型以及不同的視圖和控制器。如需詳細資訊，請參閱 Spring Boot。

本服務使用以下 AWS 服務：

- Amazon Rekognition
- Amazon Simple Storage Service (Amazon S3)
- Amazon SES
- AWS Elastic Beanstalk

本教學課程中包含的 AWS 服務包含在 AWS 免費方案中。我們建議您在完成教學課程中建立的所有資源後終止這些資源，以避免收費。

必要條件

在開始之前，您需要完成設定 Java AWS SDK 中的步驟。請確定執行下列操作：

- Java 1.8 JDK.
- Maven 3.6 或更高版本。
- 一個名為影片 [某個值] 的 Amazon S3 儲存貯體。請務必在您的 Amazon S3 Java 程式碼中使用此儲存貯體名稱。如需詳細資訊，請參閱建立儲存貯體。
- IAM 角色。您將要創建的VideoDetectFaces類需要此功能。如需詳細資訊，請參閱設定 Amazon Rekognition Video。
- 有效的 Amazon SNS 主題。您將要創建的VideoDetectFaces類需要此功能。如需詳細資訊，請參閱設定 Amazon Rekognition Video。

程序

在本教學課程中，您要執行以下作業：

1. 建立專案
2. 將 POM 相依性新增到您的專案
3. 建立 Java 類別
4. 建立 HTML 檔案
5. 建立指令碼檔案
6. 將專案打包至 JAR 檔案
7. 將應用程式部署到 AWS Elastic Beanstalk

若要繼續進行教學課程，請遵循AWS 文件 SDK 範例 GitHub 儲存庫中的詳細指示。

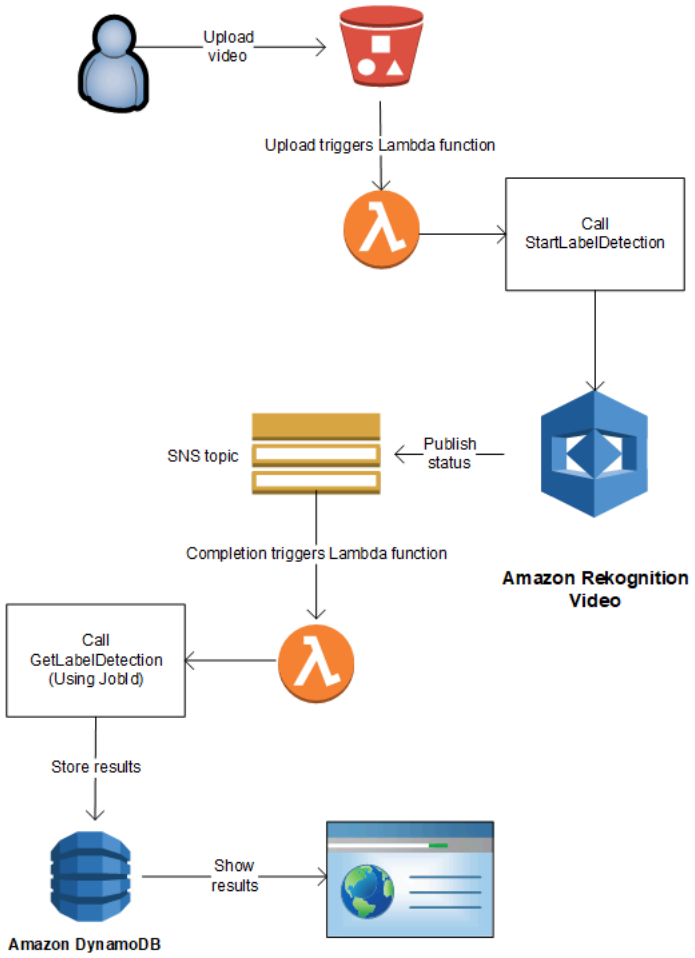
建立 Amazon Rekognition Lambda 函數

本教學說明如何使用 Java Lambda 函數，取得標籤偵測影片分析操作的結果。

> **Note**
>
> 本教學課程使用 AWS SDK for Java 1.x。如需使用 Rekognition 和 Java 第 2 版 AWS SDK 的教學課程，請參閱AWS 文件 SDK 範例儲存庫。 GitHub

您可以將 Lambda 函數與 Amazon Rekognition Video 操作搭配使用。例如，以下圖表顯示網站使用 Lambda 函數，在影片上傳到 Amazon S3 儲存貯體時自動開始分析影片。觸發 Lambda 函數時，它會呼叫StartLabelDetection開始偵測上傳視訊中的標籤。如需使用 Lambda 處理來自 Amazon S3 儲存貯體事件通知的相關資訊，請參閱搭配 Amazon S3 事件使用 AWS Lambda。

當分析完成狀態傳送到註冊的 Amazon SNS 主題時，第二個 Lambda 函數就會觸發。第二個 Lambda 函數會呼叫GetLabelDetection以取得分析結果。然後，結果會存放在資料庫以準備顯示於網頁。本教學的重點在於此第二個 lambda 函數。



在本教學中，當 Amazon Rekognition Video 將影片分析完成狀態傳送到註冊的 Amazon SNS 主題時，Lambda 函數就會觸發。然後，它通過調用收集視頻分析結果GetLabelDetection。為了展示目的，本自學課程會將標示偵測結果寫入 CloudWatch 記錄。在您應用程式中的 Lambda 函數，您應該儲存分析結果以供日後使用。例如，您可以使用 Amazon DynamoDB 以儲存分析結果。如需詳細資訊，請參閱使用 DynamoDB。

下列程序將告訴您如何做到：

- 建立 Amazon SNS 主題和設定許可。
- 使用建立 Lambda 函數， AWS Management Console 並將其訂閱至 Amazon SNS 主題。
- 使用 AWS Management Console設定 Lambda 函數。
- 將範例程式碼新增至 AWS Toolkit for Eclipse 專案，並將其上傳至 Lambda 函數。
- 使用 AWS CLI測試 Lambda 函數。

> **Note**
>
> 在整個自學課程中使用相同的 AWS 區域。

必要條件

本教學假設您已熟悉 AWS Toolkit for Eclipse。如需詳細資訊，請參閱 AWS Toolkit for Eclipse。

建立 SNS 主題：

Amazon Rekognition Video 影片分析操作完成狀態已傳送至 Amazon SNS 主題。此程序會建立 Amazon SNS 主題和 IAM 服務角色，讓 Amazon Rekognition Video 存取您的 Amazon SNS 主題。如需詳細資訊，請參閱 呼叫 Amazon Rekognition Video 操作。

建立 Amazon SNS 主題

1. 如果您尚未建立 IAM 服務角色，請先建立才可對該角色提供存取 Amazon SNS 主題的 Amazon Rekognition Video 權限。請記下 Amazon Resource Name (ARN)。如需詳細資訊，請參閱 提供對多個 Amazon SNS 主題的存取權限。
2. 使用 Amazon SNS 主控台建立 Amazon SNS 主題。您只需要指定主題名稱即可。在主題名稱前面加上。AmazonRekognition請記下主題 ARN。

建立 Lambda 函式

您使用 AWS Management Console建立 Lambda 函數。然後，您使用 AWS Toolkit for Eclipse 專案將 Lambda 函數套件上傳 AWS Lambda。您也可以使用 AWS Toolkit for Eclipse建立 Lambda 函數。如需詳細資訊，請參閱教學：如何建立、上傳和叫用 AWS Lambda 函數。

建立 Lambda 函數

1. 簽署 AWS 管理主控台，並開啟位於 https://console.aws.amazon.com/lambda/ 的 AWS Lambda 主控台。
2. 選擇建立函數。
3. 選擇從頭開始撰寫。
4. 在 Function Name (函數名稱) 中，為您的函數輸入名稱。
5. 在 Runtime* (執行時間*) 中，選擇 Java 8。

6. 選擇 Choose or create an execution role (選擇或建立執行角色)。
7. 在 Execution role (執行角色) 中，選擇 Create a new role with basic Lambda permissions (建立具備基本 Lambda 許可的新角色)。
8. 請注意，新角色名稱會顯示在 Basic information (基本資訊) 區段的底部。
9. 選擇建立函數。

設定 Lambda 函數

建立 Lambda 函數後，您可以設定它來讓您在 建立 SNS 主題： 中建立的 Amazon SNS 主題予以觸發。您也將調整 Lambda 函數的記憶體需求和逾時期間。

設定和部署 Lambda 函數

1. 在 Function Code (函數程式碼) 的 `com.amazonaws.lambda.demo.JobCompletionHandlerHandler` (處理常式) 中輸入 。
2. 在 Basic settings (基本設定) 中，選擇 Edit (編輯)。Edit basic settings (編輯基本設定) 對話方塊隨即顯示。

   a. 在記憶體中選擇 1024。
   b. 在 Timeout (逾時) 中選擇 10 秒。
   c. 選擇儲存。
3. 在 Designer (設計工具) 中，選擇 +Add trigger (+新增觸發條件)。[Add trigger (新增觸發條件)] 對話方塊隨即顯示。
4. 在 Trigger configuration (觸發條件組態) 中選擇 SNS。

   在 SNS 主題中，選擇您在 建立 SNS 主題： 中建立的 Amazon SNS 主題。
5. 選擇 Enable trigger (啟用觸發條件)。
6. 若要新增觸發條件，請選擇 Add (新增)。
7. 選擇儲存，以儲存更新的 Lambda 函數。

設定 IAM Lambda 角色

若要呼叫 Amazon Rekognition Video 操作，您可以將 AmazonRekognitionFullAccessAWS 受管政策新增至 IAM Lambda 角色。開始操作 (例如 StartLabelDetection) 也需要 Amazon Rekognition 影片用來存取 Amazon SNS 主題的 IAM 服務角色傳遞角色許可。

設定角色

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 在導覽窗格中，選擇角色。
3. 在清單中，選擇您在 建立 Lambda 函式 中建立之執行角色的名稱。
4. 選擇許可索引標籤標籤。
5. 選擇連接政策。
6. AmazonRekognitionFullAccess從策略清單中選擇。
7. 選擇連接政策。
8. 再次選擇執行角色。
9. 選擇 Add inline policy (新增內嵌政策)。
10. 選擇 JSON 標籤。
11. 用以下政策取代現有政策。用您在 建立 SNS 主題： 中建立的 IAM 服務角色取代 `servicerole`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "mysid",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:servicerole"
        }
    ]
}
```

12. 選擇檢閱政策。
13. 在 Name* (名稱*) 中輸入政策的名稱。
14. 選擇建立政策。

建立 AWS Toolkit for Eclipse Lambda 專案

觸發 Lambda 函數時，下列程式碼會從 Amazon SNS 主題取得完成狀態，並呼叫GetLabelDetection以取得分析結果。偵測到的標籤計數，以及偵測到的標籤清單會寫入 CloudWatch 記錄。您的 Lambda 函數應儲存影片分析結果以供日後使用。

若要建立 L AWS Toolkit for Eclipse ambda 專案

1. 建立一個 AWS Toolkit for EclipseAWS Lambda 專案。
   - 在 Project name: (專案名稱：) 中，輸入您選擇的專案名稱。
   - 對於類別名稱:，輸入JobCompletionHandler。
   - 在 Input type: (輸入類型：) 中選擇 SNS Event (SNS 事件)。
   - 保留其他欄位不變。
2. 在 Eclipse 專案資源管理器中，開啟產生的 Lambda 處理常式方法 (JobCompletionHandler.java)，並使用下列命令取代內容：

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.lambda.demo;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import java.util.List;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
import com.amazonaws.services.rekognition.model.LabelDetection;
import com.amazonaws.services.rekognition.model.LabelDetectionSortBy;
import com.amazonaws.services.rekognition.model.VideoMetadata;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
public class JobCompletionHandler implements RequestHandler<SNSEvent, String> {

   @Override
   public String handleRequest(SNSEvent event, Context context) {

      String message = event.getRecords().get(0).getSNS().getMessage();
      LambdaLogger logger = context.getLogger();

      // Parse SNS event for analysis results. Log results
      try {
         ObjectMapper operationResultMapper = new ObjectMapper();
         JsonNode jsonResultTree = operationResultMapper.readTree(message);
         logger.log("Rekognition Video Operation:========================");
         logger.log("Job id: " + jsonResultTree.get("JobId"));
         logger.log("Status : " + jsonResultTree.get("Status"));
         logger.log("Job tag : " + jsonResultTree.get("JobTag"));
         logger.log("Operation : " + jsonResultTree.get("API"));

         if (jsonResultTree.get("API").asText().equals("StartLabelDetection")) {

            if (jsonResultTree.get("Status").asText().equals("SUCCEEDED")){
               GetResultsLabels(jsonResultTree.get("JobId").asText(), context);
            }
            else{
               String errorMessage = "Video analysis failed for job "
                     + jsonResultTree.get("JobId")
                     + "State " + jsonResultTree.get("Status");
               throw new Exception(errorMessage);
            }

         } else
            logger.log("Operation not StartLabelDetection");

      } catch (Exception e) {
         logger.log("Error: " + e.getMessage());
         throw new RuntimeException (e);


      }

      return message;
   }

   void GetResultsLabels(String startJobId, Context context) throws Exception {
```

```
     LambdaLogger logger = context.getLogger();

     AmazonRekognition rek =
AmazonRekognitionClientBuilder.standard().withRegion(Regions.US_EAST_1).build();

     int maxResults = 1000;
     String paginationToken = null;
     GetLabelDetectionResult labelDetectionResult = null;
     String labels = "";
     Integer labelsCount = 0;
     String label = "";
     String currentLabel = "";

     //Get label detection results and log them.
     do {

        GetLabelDetectionRequest labelDetectionRequest = new
GetLabelDetectionRequest().withJobId(startJobId)

.withSortBy(LabelDetectionSortBy.NAME).withMaxResults(maxResults).withNextToken(pagination

        labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

        paginationToken = labelDetectionResult.getNextToken();
        VideoMetadata videoMetaData = labelDetectionResult.getVideoMetadata();

        // Add labels to log
        List<LabelDetection> detectedLabels = labelDetectionResult.getLabels();

        for (LabelDetection detectedLabel : detectedLabels) {
           label = detectedLabel.getLabel().getName();
           if (label.equals(currentLabel)) {
              continue;
           }
           labels = labels + label + " / ";
           currentLabel = label;
           labelsCount++;

        }
     } while (labelDetectionResult != null &&
labelDetectionResult.getNextToken() != null);

     logger.log("Total number of labels : " + labelsCount);
```

```
        logger.log("labels : " + labels);

    }


}
```

3. Rekognition 命名空間未解析。若要修正此問題：
   - 將滑鼠暫停在 `import com.amazonaws.services.rekognition.AmazonRekognition;` 這行上有底線的部分。
   - 選擇 Fix project set up... (修復專案設定...)。
   - 選擇 Amazon Rekognition存檔的最新版本。
   - 選擇 OK 以新增存檔到專案。
4. 儲存檔案。
5. 在 Eclipse 程式碼視窗中按一下滑鼠右鍵，選擇 AWS Lambda，然後選擇 Upload function to AWS Lambda (將函數上傳至 AWS Lambda)。
6. 在 Select Target Lambda Function (選擇目標 Lambda 函數) 頁面中，選擇要使用的 AWS 區域。
7. 選擇 Choose an existing lambda function (選擇現有的 Lambda 函數)，然後選取您在 建立 Lambda 函式 中建立的 Lambda 函數。
8. 選擇下一步。Function Configuration (功能組態) 對話方塊隨即顯示。
9. 在 IAM role (IAM 角色) 中選擇您在 建立 Lambda 函式 建立的 IAM 角色。
10. 選擇完成，Lambda 函數就會上傳至 AWS。

測試 Lambda 函數

透過啟動視訊的標籤偵測分析，使用下列 AWS CLI 命令來測試 Lambda 函數。在分析完成後，就會觸發 Lambda 函數。檢查 CloudWatch 記錄檔，確認分析成功。

測試 Lambda 函數

1. 將 MOV 或 MPEG-4 格式的影片上傳到 S3 儲存貯體。為達測試目的，請上傳長度不超過 30 秒的影片。

如需指示說明，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件至 Amazon S3。

2. 執行下列 AWS CLI 命令以開始偵測視訊中的標籤。

```
aws rekognition start-label-detection --video
 "S3Object={Bucket="bucketname",Name="videofile"}" \
--notification-channel "SNSTopicArn=TopicARN,RoleArn=RoleARN" \
--region Region
```

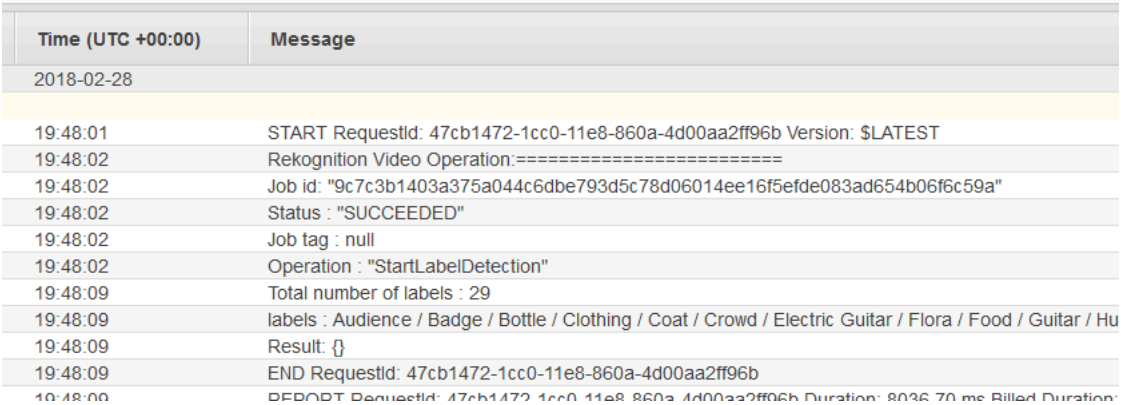
更新下列的值：

- 將 bucketname 和 videofile 變更為 Amazon S3 儲存貯體名稱和您想要偵測標籤的影片檔案名稱。
- 將 TopicARN 變更為您在 建立 SNS 主題： 中建立的 Amazon SNS 主題的 ARN。
- 將 RoleARN 變更為您在 建立 SNS 主題： 中建立的 IAM 角色的 ARN。
- 變更Region為您正在使用的 AWS 區域。

3. 請記下回應中的 JobId 值。回應看起來類似以下 JSON 範例。

```
{
    "JobId": "547089ce5b9a8a0e7831afa655f42e5d7b5c838553f1a584bf350ennnnnnnnnn"
}
```

4. 開啟主控台，網址為：https://console.aws.amazon.com/cloudwatch/。
5. 當分析完成後，Lambda 函數的日誌專案會顯示在日誌群組。
6. 選擇 Lambda 函數以查看日誌串流。
7. 選擇最新的日誌串流，以查看 Lambda 函數建立的日誌專案。如果操作成功，它看起來類似於以下輸出，該輸出顯示視頻識別操作的詳細信息，包括作業 ID，操作類型 StartLabelDetection「」以及檢測到的標籤類別（例如瓶子，服裝，人群和食物）的列表：



Time (UTC +00:00)	Message
2018-02-28	
19:48:01	START RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b Version: $LATEST
19:48:02	Rekognition Video Operation:==========
19:48:02	Job id: "9c7c3b1403a375a044c6dbe793d5c78d06014ee16f5efde083ad654b06f6c59a"
19:48:02	Status : "SUCCEEDED"
19:48:02	Job tag : null
19:48:02	Operation : "StartLabelDetection"
19:48:09	Total number of labels : 29
19:48:09	labels : Audience / Badge / Bottle / Clothing / Coat / Crowd / Electric Guitar / Flora / Food / Guitar / Hu
19:48:09	Result: {}
19:48:09	END RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b
19:48:09	REPORT RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b Duration: 8036.70 ms Billed Duration:

ID 的值應符合您在步驟 3 記下的 `JobId` 值。

使用 Amazon 重新認知進行身分驗證

Amazon Rekognition 為使用者提供多項操作，可讓您輕鬆建立身分驗證系統。Amazon Rekognition 可讓使用者偵測影像中的臉孔，然後比較臉部資料，將任何偵測到的臉孔與其他臉孔進行比較。此臉部數據存儲在稱為集合的服務器端容器中。透過使用 Amazon Rekognition 的臉部偵測、臉部比較和集合管理操作，您可以使用身分驗證解決方案建立應用程式。

本教學課程將示範兩種常見的工作流程，以建立需要身分驗證的應用程式。

第一個工作流程涉及在集合中註冊新使用者。第二個工作流程涉及為了登錄返回的用戶而搜索現有集合。

您將使用AWS適用於蟒蛇的 SDK對於本教程。您還可以看到AWS說明文件 SDK 範例GitHub回購有關更多 Python 教程。

主題

- 先決條件
- 建立集合
- 新用戶註冊
- 現有使用者登入

先決條件

在開始本教學課程之前，您將需要安裝 Python 並完成所需的步驟設置蟒蛇AWSSDK。除此之外，請確保您擁有：

- 創建了一個AWS帳戶和 IAM 角色
- 安裝了 Python 軟件開發套件
- 正確配置您的AWS存取認證
- 創建了一個亞馬遜簡單存儲服務桶並上傳了您希望用作身份驗證的圖像。
- 選擇第二張影像作為身份驗證的目標影像。

建立集合

您必須先擁有要使用的集合，才能在集合中註冊新使用者或搜尋系列。Amazon Rekognition 系列是伺服器端容器，用於儲存偵測到的臉孔的相關資訊。

建立 集合

您將首先編寫一個函數，該函數會建立供應用程式使用的集合。Amazon Rekognition 會在伺服器端容器 (稱為集合) 中儲存偵測到的臉孔相關資訊。您可以搜尋儲存在「集合」中的臉部資訊，找出已知的臉孔。若要儲存臉部資訊，您首先需要使用`CreateCollection`操作。

1. 選取您要建立的收藏名稱。在下列程式碼中，取代的值`collection_id`使用您要創建的集合的名稱，並替換`region`使用您的用戶憑據中定義的區域名稱。您可以使用`Tags`參數將您想要的任何標籤應用於集合，儘管這不是必需的。該`CreateCollection`操作將返回有關您創建的集合的信息，包括集合的 Arn。記下您在執行程式碼時收到的 Arn。

```
import boto3

def create_collection(collection_id, region):
    client = boto3.client('rekognition', region_name=region)

    # Create a collection
    print('Creating collection:' + collection_id)
    response = client.create_collection(CollectionId=collection_id,
    Tags={"SampleKey1":"SampleValue1"})
    print('Collection ARN: ' + response['CollectionArn'])
    print('Status code: ' + str(response['StatusCode']))
    print('Done...')

collection_id = 'collection-id-name'
region = "region-name"
create_collection(collection_id, region)
```

2. 保存並運行代碼。複製下來集合阿恩。

   現在 Rekognition 系列已經建立完畢，您可以將臉部資訊和識別碼儲存在該集合中。您還可以將人臉與存儲的信息進行比較以進行驗證。

新用戶註冊

您將希望能夠註冊新用戶並將他們的信息添加到收藏中。註冊新用戶的過程通常包括以下步驟：

呼叫**DetectFaces**操作

編寫代碼以通過以下方式檢查臉部圖像的質量DetectFaces操作。您將使用DetectFaces操作，以確定由相機捕獲的圖像是否適合由SearchFacesByImage操作。影像應該只包含一個臉孔。您將提供一個本地輸入圖像文件DetectFaces操作和接收圖像中檢測到的面孔的詳細信息。下面的示例代碼提供了輸入圖像DetectFaces然後檢查影像中是否偵測到只有一個臉孔。

1. 在下列程式碼範例中，取代photo使用您要在其中檢測臉部的目標圖像的名稱。您還需要替換的值region使用與您的帳戶關聯的地區名稱。

```
import boto3
import json

def detect_faces(target_file, region):

    client=boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.detect_faces(Image={'Bytes': imageTarget.read()},
    Attributes=['ALL'])

    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')

        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))

        # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
```

```
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])

photo = 'photo-name'
region = 'region-name'
face_count=detect_faces(photo, region)
print("Faces detected: " + str(face_count))

if face_count == 1:
    print("Image suitable for use in collection.")
else:
    print("Please submit an image with only one face.")
```

2. 保存並運行正在進行的代碼。

呼叫**CompareFaces**操作

您的應用程式必須能夠在集合中註冊新使用者，並確認回訪使用者的身分。您將首先創建用於註冊新用戶的功能。您將開始使用CompareFaces用於比較用戶的本地輸入/目標圖像和 ID/ 存儲圖像的操作。如果在兩個影像中偵測到的臉孔之間存在相符項目，您可以搜尋集合，以查看使用者是否已在其中註冊。

首先撰寫一個函數，將輸入影像與您存放在 Amazon S3 儲存貯體中的 ID 映像進行比較。在下面的代碼示例中，您將需要自己提供輸入圖像，這些圖像應在使用某種形式的活性檢測器後被捕獲。您還需要傳遞存放在 Amazon S3 儲存貯體中的映像名稱。

1. 替換的值bucket使用包含來源檔案的 Amazon S3 儲存貯體的名稱。您還需要替換的值source_file使用您正在使用的源圖像的名稱。替換的值target_file使用您提供的目標文件的名稱。替換的值region使用的名稱region在您的使用者認證中定義。

   您也需要在回應中傳回的相符項目中指定最小信賴等級，使用similarityThreshold參數。偵測到的臉孔只會傳回FaceMatches陣列，如果置信度高於此閾值。您選擇的similarityThreshold應反映您特定使用案例的性質。任何涉及重要安全性應用程式的使用案例都應使用 99 做為選取的閾值。

   ```
   import boto3

   def compare_faces(bucket, sourceFile, targetFile, region):
   ```

```
    client = boto3.client('rekognition', region_name=region)

    imageTarget = open(targetFile, 'rb')

    response = client.compare_faces(SimilarityThreshold=99,
                                    SourceImage={'S3Object':
{'Bucket':bucket,'Name':sourceFile}},
                                    TargetImage={'Bytes': imageTarget.read()})

    for faceMatch in response['FaceMatches']:
        position = faceMatch['Face']['BoundingBox']
        similarity = str(faceMatch['Similarity'])
        print('The face at ' +
              str(position['Left']) + ' ' +
              str(position['Top']) +
              ' matches with ' + similarity + '% confidence')

    imageTarget.close()
    return len(response['FaceMatches'])

bucket = 'bucket-name'
source_file = 'source-file-name'
target_file = 'target-file-name'
region = "region-name"
face_matches = compare_faces(bucket, source_file, target_file, region)
print("Face matches: " + str(face_matches))

if str(face_matches) == "1":
    print("Face match found.")
else:
    print("No face match found.")
```

2. 保存並運行正在進行的代碼。

   將返回一個響應對象，其中包含有關匹配的面和可信度級別的信息。

呼叫**SearchFacesByImage**操作

如果置信水平CompareFaces操作高於您選擇的SimilarityThreshold時，您需要在收藏中搜尋可能與輸入影像相符的臉孔。如果在您的收藏中找到匹配項，這意味著該用戶很可能已經在收藏中註冊，並且不需要在您的收藏中註冊新用戶。如果沒有相符項目，您可以在收藏中註冊新使用者。

1. 首先編寫將調用SearchFacesByImage操作。該操作將以本地圖像文件作為參數，然後搜索您的Collection表示符合所提供影像中偵測到的最大臉孔的臉孔。

   在下列程式碼範例中，變更的值collectionId到您要搜索的收藏。替換的值region使用與您的帳戶關聯的地區名稱。您還需要替換的值photo使用您的輸入文件的名稱。您還需要通過替換的值來指定相似性閾值threshold與選擇的百分位數。

   ```
   import boto3

   collectionId = 'collection-id-name'
   region = "region-name"
   photo = 'photo-name'
   threshold = 99
   maxFaces = 1
   client = boto3.client('rekognition', region_name=region)

   # input image should be local file here, not s3 file
   with open(photo, 'rb') as image:
       response = client.search_faces_by_image(CollectionId=collectionId,
       Image={'Bytes': image.read()},
       FaceMatchThreshold=threshold, MaxFaces=maxFaces)

   faceMatches = response['FaceMatches']
   print(faceMatches)

   for match in faceMatches:
       print('FaceId:' + match['Face']['FaceId'])
       print('ImageId:' + match['Face']['ImageId'])
       print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
       print('Confidence: ' + str(match['Face']['Confidence']))
   ```

2. 保存並運行正在進行的代碼。如果有匹配項，則表示圖像中識別的人已經是收藏的一部分，因此無需繼續進行後續步驟。在這種情況下，您可以只允許用戶訪問應用程序。

呼叫**IndexFaces**操作

假設在您搜尋的集合中找不到相符項目，您會想要將使用者的臉孔新增至您的收藏。您可以通過調用IndexFaces操作。當你打電話IndexFacesAmazon Rekognition 會擷取輸入影像中識別的臉孔的臉部特徵，並將資料儲存在指定的集合中。

1. 首先編寫代碼以調用IndexFaces。替換的值image使用您要用作輸入圖像的本地文件的名稱IndexFaces操作。您還需要替換的值photo_name使用所需的輸入圖像名稱。一定要替換的值collection_id使用您先前建立的集合 ID。接下來，替換的值region使用與您的帳戶關聯的地區名稱。您還需要指定一個值MaxFacesinput 參數，用於定義影像中應建立索引的最大面孔數目。此參數的預設值為 1。

```
import boto3

def add_faces_to_collection(target_file, photo, collection_id, region):
    client = boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.index_faces(CollectionId=collection_id,
                                  Image={'Bytes': imageTarget.read()},
                                  ExternalImageId=photo,
                                  MaxFaces=1,
                                  QualityFilter="AUTO",
                                  DetectionAttributes=['ALL'])
    print(response)

    print('Results for ' + photo)
    print('Faces indexed:')
    for faceRecord in response['FaceRecords']:
        print('  Face ID: ' + faceRecord['Face']['FaceId'])
        print('  Location: {}'.format(faceRecord['Face']['BoundingBox']))
        print('  Image ID: {}'.format(faceRecord['Face']['ImageId']))
        print('  External Image ID: {}'.format(faceRecord['Face']
['ExternalImageId']))
        print('  Confidence: {}'.format(faceRecord['Face']['Confidence']))

    print('Faces not indexed:')
    for unindexedFace in response['UnindexedFaces']:
        print(' Location: {}'.format(unindexedFace['FaceDetail']['BoundingBox']))
        print(' Reasons:')
        for reason in unindexedFace['Reasons']:
            print('   ' + reason)
    return len(response['FaceRecords'])

image = 'image-file-name'
collection_id = 'collection-id-name'
photo_name = 'desired-image-name'
region = "region-name"
```

```
indexed_faces_count = add_faces_to_collection(image, photo_name, collection_id,
 region)
print("Faces indexed count: " + str(indexed_faces_count))
```

2. 保存並運行正在進行的代碼。判斷您是否要儲存任何傳回的資料IndexFaces作業，例如指定給影像中人物的 FaceID。下一節將研究如何保存這些數據。複製下來返回FaceId,ImageId，以及Confidence繼續之前的值。

將影像和臉部識別碼資料存放在亞馬遜 S3 和亞馬遜

取得輸入影像的臉部識別碼後，影像資料即可儲存在 Amazon S3 中，而臉部資料和影像 URL 則可輸入 DynamoDB 等資料庫中。

1. 撰寫程式碼，將輸入影像上傳至 Amazon S3 資料庫。在接下來的程式碼範例中，取代bucket使用您要上傳文件的存儲桶的名稱，然後替換file_name使用您想要存放在 Amazon S3 儲存貯體中的本機檔案名稱。提供金鑰名稱，以取代下列值來識別 Amazon S3 儲存貯體中的檔案key_name使用您想要提供圖像文件的名稱。您要上傳的檔案與先前程式碼範例中定義的檔案相同，也就是您使用的輸入檔案IndexFaces。最後，替換的值region使用與您的帳戶關聯的地區名稱。

```
import boto3
import logging
from botocore.exceptions import ClientError

# store local file in S3 bucket
bucket = "bucket-name"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
s3 = boto3.client('s3', region_name=region)
# Upload the file
try:
    response = s3.upload_file(file_name, bucket, key_name)
    print("File upload successful!")
except ClientError as e:
    logging.error(e)
```

2. 儲存並執行正在進行的程式碼範例，將您的輸入影像上傳到 Amazon S3。

3. 您也需要將返回的 Face ID 保存到數據庫中。這可以透過建立 DynamoDB 資料庫表格，然後將臉部識別碼上傳至該資料表來完成。下列程式碼範例會建立 DynamoDB 資料表。請注意，您只需要運行一次創建此表的代碼。在下列程式碼中，取代的值`region`與您的帳戶關聯的地區的價值。您還需要替換的值`database_name`使用您想要為 DynamoDB 表格提供的名稱。

```
import boto3

# Create DynamoDB database with image URL and face data, face ID

def create_dynamodb_table(table_name, region):
    dynamodb = boto3.client("dynamodb", region_name=region)

    table = dynamodb.create_table(
        TableName=table_name,
        KeySchema=[{
                'AttributeName': 'FaceID', 'KeyType': 'HASH'  # Partition key
                },],
            AttributeDefinitions=[
            {
                'AttributeName': 'FaceID', 'AttributeType': 'S'  }, ],
                ProvisionedThroughput={
            'ReadCapacityUnits': 10, 'WriteCapacityUnits': 10  }
    )
    print(table)
    return table

region = "region-name"
database_name = 'database-name'
dynamodb_table = create_dynamodb_table(database_name, region)
print("Table status:", dynamodb_table)
```

4. 保存並運行正在進行的代碼以創建表格。

5. 創建表後，您可以上傳返回的FaceId到它。若要這麼做，您將使用 Table 函數建立到資料表的連線，然後使用`put_item`功能上傳數據。

   在下列程式碼範例中，取代`bucket`使用儲存貯體的名稱，其中包含您上傳到 Amazon S3 的輸入影像。您還需要替換的值 `file_name`使用您上傳到 Amazon S3 儲存貯體的輸入檔案名稱，以及`key_name`使用您之前用來識別輸入文件的密鑰。最後，替換的值`region`使用與您的帳戶關聯的地區名稱。這些值應與步驟 1 中提供的值相符。

該AddDBEntry存儲FaceId,ImageId，以及指定給集合中面的信賴度值。提供以下功能，其中包含您在繼續步驟 2 中儲存的值IndexFaces部分。

```
import boto3
from pprint import pprint
from decimal import Decimal
import json

# The local file that was stored in S3 bucket
bucket = "s3-bucket-name"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
# Get URL of file
file_url = "https://s3.amazonaws.com/{}/{}".format(bucket, key_name)
print(file_url)

# upload face-id, face info, and image url
def AddDBEntry(file_name, file_url, face_id, image_id, confidence):
    dynamodb = boto3.resource('dynamodb', region_name=region)
    table = dynamodb.Table('FacesDB-4')
    response = table.put_item(
       Item={
            'ExternalImageID': file_name,
            'ImageURL': file_url,
            'FaceID': face_id,
            'ImageID': image_id,
            'Confidence': json.loads(json.dumps(confidence), parse_float=Decimal)
       }
    )
    return response

# Mock values for face ID, image ID, and confidence - replace them with actual
 values from your collection results
dynamodb_resp = AddDBEntry(file_name, file_url, "FACE-ID-HERE",
    "IMAGE-ID-HERE", confidence-here)
print("Database entry successful.")
pprint(dynamodb_resp, sort_dicts=False)
```

6. 儲存並執行正在進行的程式碼範例，將傳回的 Face ID 資料儲存在資料表中。

現有使用者登入

使用者在集合中註冊後，可以使用SearchFacesByImage操作。您將需要獲取輸入圖像，然後使用檢查輸入圖像的質量DetectFaces。這確定在運行之前是否已使用合適的圖像SearchFacesbyImage操作。

呼叫DetectFaces操作

1. 您將使用DetectFaces檢查臉部圖像質量的操作，並確定由相機捕獲的圖像是否適合由SearchFacesByImage操作。輸入圖像應該只包含一張臉。下列程式碼範例會取得輸入影像，並將其提供給DetectFaces操作。

   在下列程式碼範例中，取代photo使用本地目標圖像的名稱並替換值region使用與您的帳戶關聯的地區名稱。

```
import boto3
import json

def detect_faces(target_file, region):

    client=boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.detect_faces(Image={'Bytes': imageTarget.read()},
    Attributes=['ALL'])

    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')

        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))

        # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])
```

```
photo = 'photo-name'
region = 'region-name'
face_count=detect_faces(photo, region)
print("Faces detected: " + str(face_count))

if face_count == 1:
    print("Image suitable for use in collection.")
else:
    print("Please submit an image with only one face.")
```

2. 保存並運行代碼。

呼叫SearchFacesByImage操作

1. 撰寫程式碼，將偵測到的臉孔與集合中的臉孔進行比較SearchFacesByImage。您將使用「新用戶註冊」部分中顯示的代碼，並將輸入圖像提供給SearchFacesByImage操作。

   在下列程式碼範例中，變更的值collectionId至您要搜尋的商品系列。您還將更改的值bucket到一個亞馬遜 S3 存儲桶的名稱和值fileName到該存儲桶中的圖像文件。替換的值region使用與您的帳戶關聯的地區名稱。您還需要通過替換的值來指定相似性閾值threshold與選擇的百分位數。

   ```
   import boto3

   bucket = 'bucket-name'
   collectionId = 'collection-id-name'
   region = "region-name"
   fileName = 'file-name'
   threshold = 70
   maxFaces = 1
   client = boto3.client('rekognition', region_name=region)

   # input image should be local file here, not s3 file
   with open(fileName, 'rb') as image:
       response = client.search_faces_by_image(CollectionId=collectionId,
       Image={'Bytes': image.read()},
       FaceMatchThreshold=threshold, MaxFaces=maxFaces)
   ```

2. 保存並運行代碼。

檢查返回的 FaceID 和置信水平

您現在可以檢查匹配的信息FaceId通過打印出響應元素FaceId，相似性和信心屬性。

```
faceMatches = response['FaceMatches']
print(faceMatches)

for match in faceMatches:
    print('FaceId:' + match['Face']['FaceId'])
    print('ImageId:' + match['Face']['ImageId'])
    print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
    print('Confidence: ' + str(match['Face']['Confidence']))
```

使用 Lambda 和 Python 偵測影像中的標籤

AWS Lambda 是一項運算服務，可供您用來執行程式碼，無需佈建或管理伺服器。您可以從 Lambda 函數中呼叫重新認知 API 作業。以下說明顯示瞭如何在 Python 中創建一個調用 Lambda 函數`DetectLabels`。

拉姆達函數調用`DetectLabels`並返回在圖像中檢測到的標籤數組，以及檢測到它們的可信度水平。

這些指示包括範例 Python 程式碼，其中示範如何呼叫 Lambda 函數，並提供來自 Amazon S3 儲存貯體或本機電腦的映像檔。

請確認您選擇的影像符合 Rekognition 的限制。請參閱指引及配額在重新認知和DetectLabelsAPI 參考資料以取得有關影像檔案類型和大小限制的資訊。

建立 Lambda 函數 (主控台)

在此步驟中，您會建立空白的 Lambda 函數和 IAM 執行角色，讓 Lambda 函數呼叫`DetectLabels`操作。在稍後的步驟中，您可以新增原始程式碼，並選擇性地將圖層新增至 Lambda 函數。

如果您使用存放在 Amazon S3 儲存貯體中的文件，此步驟也會示範如何授予存放文件儲存貯體的存取權。

若要建立AWS Lambda功能（控制台）

1. 請登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/lambda/ 的 AWS Lambda 主控台。
2. 選擇 建立函數 。如需詳細資訊，請參閱使用主控台建立 Lambda 函數。

3. 選擇下列選項：

   - 選擇 從頭開始撰寫。
   - 輸入以下項目的值函數名稱。
   - 對於运行时，選擇最新版本的 Python。
   - 對於 Architecture (架構)，選擇 x86_64。

4. 選擇建立函數以建立AWS Lambda功能。
5. 在功能頁面上，選擇配置標籤。
6. 在「」權限窗格的下方執行角色」下方，選擇要在 IAM 主控台中開啟角色的角色名稱。
7. 在權限」頁籤上，選擇新增權限然後建立內嵌政策。
8. 選擇合適的JSON索引標籤，並以下列原則取代原則：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "rekognition:DetectLabels",
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "DetectLabels"
        }
    ]
}
```

9. 選擇 Review policy (檢閱政策)。
10. 輸入策略的名稱，例如DetectLabels-訪問。
11. 選擇 建立政策。
12. 如果您要將文件存放在 Amazon S3 儲存貯體中進行分析，則必須新增 Amazon S3 存取政策。若要這麼做，請重複步驟 7 到 11AWS Lambda控制台並進行以下更改。

    a. 對於步驟 8，請使用下列原則。取代*##/#####*使用 Amazon S3 儲存貯體和您要分析之文件的資料夾路徑。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
```

```
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::bucket/folder path/*"
        }
    ]
}
```

b. 對於步驟 10，選擇不同的策略名稱，例如S3 儲存貯體存取。

(選擇性) 建立圖層 (主控台)

您不需要執行此步驟即可使用 Lambda 函數並呼叫DetectLabels。

該DetectLabels作業包含在預設的 Lambda Python 環境中，作為AWS開發套件為蟒蛇 (肉毒 3).

如果您的 Lambda 函數的其他部分需要最近的AWS不在預設 Lambda Python 環境中的服務更新，然後您可以執行此步驟，將最新的 Boto3 SDK 版本新增為函數的一個層。

若要將 SDK 新增為圖層，您必須先建立包含 Boto3 SDK 的壓縮檔案歸檔。然後，創建一個圖層並將 zip 文件存檔添加到圖層中。如需詳細資訊，請參閱搭配 Lambda 函數使用圖層。

建立和新增圖層 (控制台) 的步驟

1. 開啟命令提示字元並輸入下列命令，以建立具有最新版本的部署套件AWSSDK。

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

2. 請注意您在此程序的步驟 8 中使用的壓縮檔案 (boto3-layer.zip) 的名稱。
3. 在 https://console.aws.amazon.com/lambda/ 開啟 AWS Lambda 主控台。
4. 在導覽窗格中，選擇 Layers (層)。
5. 選擇 Create layer (建立 Layer)。
6. 輸入 Name (名稱) 與 Description (描述) 的值。
7. 對於密碼輸入類型，選擇上傳一個 .zip 文件並選擇上传。
8. 在對話方塊中，選擇您在此程序的步驟 1 中建立的壓縮檔案封存 (boto3-layer.zip)。
9. 對於相容的執行階段，選擇最新版本的 Python。
10. 選擇創建以建立圖層。
11. 選擇導覽窗格功能表圖示。

12. 在導覽窗格中，選擇函數。
13. 在資源清單中，選擇您先前在其中建立的函數???。
14. 選取程式碼索引標籤。
15. 在圖層區段中，選擇新增圖層。
16. 選擇自訂圖層。
17. 在自訂圖層，選擇您在步驟 6 中輸入的圖層名稱。
18. 在版本選擇圖層版本，應為 1。
19. 選擇 Add (新增)。

添加 Python 代碼（控制台）

在此步驟中，您可以透過 Lambda 主控台程式碼編輯器將 Python 程式碼新增至 Lambda 函數。程式碼會使用`DetectLabels`操作。它返回在圖像中檢測到的標籤的數組，以及檢測到的標籤的可信度水平。

您提供給`DetectLabels`操作可以位於 Amazon S3 儲存貯體或本機電腦中。

若要新增 Python 程式碼 (主控台)

1. 導覽至代碼標籤。
2. 在程式碼編輯器中，取代程式碼lambda_function.py使用以下代碼：

```
import boto3
import logging
from botocore.exceptions import ClientError
import json
import base64

# Instantiate logger
logger = logging.getLogger(__name__)

# connect to the Rekognition client
rekognition = boto3.client('rekognition')

def lambda_handler(event, context):

    try:
        image = None
        if 'S3Bucket' in event and 'S3Object' in event:
```

```
            s3 = boto3.resource('s3')
            s3_object = s3.Object(event['S3Bucket'], event['S3Object'])
            image = s3_object.get()['Body'].read()

        elif 'image' in event:
            image_bytes = event['image'].encode('utf-8')
            img_b64decoded = base64.b64decode(image_bytes)
            image = img_b64decoded


        elif image is None:
            raise ValueError('Missing image, check image or bucket path.')

        else:
            raise ValueError("Only base 64 encoded image bytes or S3Object are
 supported.")

        response = rekognition.detect_labels(Image={'Bytes': image})
        lambda_response = {
            "statusCode": 200,
            "body": json.dumps(response)
        }
        labels = [label['Name'] for label in response['Labels']]
        print("Labels found:")
        print(labels)

    except ClientError as client_err:

        error_message = "Couldn't analyze image: " + client_err.response['Error']
['Message']

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": client_err.response['Error']['Code'],
                "ErrorMessage": error_message
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, error_message)


    except ValueError as val_error:
```

```
        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": "ValueError",
                "ErrorMessage": format(val_error)
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, format(val_error))

    return lambda_response
```

3. 選擇部署以部署您的 Lambda 函數。

若要新增 Python 程式碼 (主控台)

現在您已經建立了 Lambda 函數，您可以叫用它來偵測影像中的標籤。

在此步驟中，您可以在電腦上執行 Python 程式碼，該程式碼會將本機映像或 Amazon S3 儲存貯體中的映像傳遞至 Lambda 函數。

確保你運行相同的代碼AWS您在其中建立 Lambda 函數的區域。您可以檢視AWSLambda 主控台中函數詳細資料頁面導覽列中 Lambda 函數的區域。

如果 Lambda 函數傳回逾時錯誤，請延長 Lambda 函數的逾時期間。如需詳細資訊，請參閱配置功能超時（控制台）。

如需有關從程式碼叫用 Lambda 函數的詳細資訊，請參閱叫用 AWS 拉姆達函數。

若要嘗試您的 Lambda 函數

1. 如果您尚未這麼做，請執行下列動作：

   a. 確保用戶具有`lambda:InvokeFunction`權限。您可以使用下列原則：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "InvokeLambda",
            "Effect": "Allow",
```

```
            "Action": "lambda:InvokeFunction",
            "Resource": "ARN for lambda function"
        }
    ]
}
```

您可以從中的函數概觀中獲取 Lambda 函數的 ARN主控台。

若要提供存取權，請新增許可到您的使用者、群組或角色：

- AWS IAM Identity Center 中的使用者和群組：

  建立許可集合。請遵循《AWS IAM Identity Center 使用者指南》的建立許可集合中的指示。

- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。請遵循《IAM 使用者指南》的為第三方身分提供者 (聯合) 建立角色中的指示。

- IAM 使用者：
  - 建立您的使用者可擔任的角色。請遵循《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接連接至使用者，或將使用者新增至使用者群組。請遵循《IAM 使用者指南》的新增許可到使用者 (主控台)中的指示。

b. 安裝和配置AWS適用於 Python 的 SDK。如需詳細資訊，請參閱步驟 2：設定 AWS CLI 和開 AWS 發套件。

2. 將以下代碼保存到名為的文件`client.py`:

```
import boto3
import json
import base64
import pprint

# Replace with the name of your S3 bucket and image object key
bucket_name = "name of bucket"
object_key = "name of file in s3 bucket"
# If using a local file, supply the file name as the value of image_path below
image_path = ""

# Create session and establish connection to client['
```

```
session = boto3.Session(profile_name='developer-role')
s3 = session.client('s3', region_name="us-east-1")
lambda_client = session.client('lambda',  region_name="us-east-1")

# Replace with the name of your Lambda function
function_name = 'RekDetectLabels'

def analyze_image_local(img_path):

    print("Analyzing local image:")

    with open(img_path, 'rb') as image_file:
        image_bytes = image_file.read()
        data = base64.b64encode(image_bytes).decode("utf8")

        lambda_payload = {"image": data}

        # Invoke the Lambda function with the event payload
        response = lambda_client.invoke(
            FunctionName=function_name,
            Payload=(json.dumps(lambda_payload))
        )

        decoded = json.loads(response['Payload'].read().decode())
        pprint.pprint(decoded)

def analyze_image_s3(bucket_name, object_key):

    print("Analyzing image in S3 bucket:")

    # Load the image data from S3 into memory
    response = s3.get_object(Bucket=bucket_name, Key=object_key)
    image_data = response['Body'].read()
    image_data = base64.b64encode(image_data).decode("utf8")

     # Create the Lambda event payload
    event = {
        'S3Bucket': bucket_name,
        'S3Object': object_key,
        'ImageBytes': image_data
    }

    # Invoke the Lambda function with the event payload
    response = lambda_client.invoke(
```

```
            FunctionName=function_name,
            InvocationType='RequestResponse',
            Payload=json.dumps(event),
            )

    decoded = json.loads(response['Payload'].read().decode())
    pprint.pprint(decoded)

def main(path_to_image, name_s3_bucket, obj_key):

    if str(path_to_image) != "":
        analyze_image_local(path_to_image)
    else:
        analyze_image_s3(name_s3_bucket, obj_key)

if __name__ == "__main__":
    main(image_path, bucket_name, object_key)
```

3. 執行程式碼。如果文件位於 Amazon S3 儲存貯體中。請確定它與之前在步驟 12 中指定的相同儲存貯體???。

   如果成功，您的程式碼會針對文件中偵測到的每個區塊類型傳回部分 JSON 回應。

使用 SDK 的 Amazon Rekognition 的程式碼範例 AWS

下列程式碼範例說明如何搭配 AWS 軟體開發套件 (SDK) 使用 Amazon Rekognition。本章中的程式碼範例旨在補充本指南其餘部分中的程式碼範例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境和跨服務範例中查看內容中的動作。

Scenarios (案例) 是向您展示如何呼叫相同服務中的多個函數來完成特定任務的程式碼範例。

Cross-service examples (跨服務範例) 是跨多個 AWS 服務執行的應用程式範例。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含入門相關資訊和舊版 SDK 的詳細資訊。

開始使用

你好 Amazon Rekognition

下列程式碼範例顯示如何開始使用 Amazon Rekognition。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

C MakeLists .txt 的 CMake 文件的代碼。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rekognition)

# Set this project's name.
```

```
project("hello_rekognition")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_rekognition.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_rekognition.cpp 來源檔案的程式碼。

```
#include <aws/core/Aws.h>
#include <aws/rekognition/RekognitionClient.h>
```

```
#include <aws/rekognition/model/ListCollectionsRequest.h>
#include <iostream>

/*
 *  A "Hello Rekognition" starter application which initializes an Amazon
 Rekognition client and
 *  lists the Amazon Rekognition collections in the current account and region.
 *
 *  main function
 *
 *  Usage: 'hello_rekognition'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Rekognition::RekognitionClient rekognitionClient(clientConfig);
        Aws::Rekognition::Model::ListCollectionsRequest request;
        Aws::Rekognition::Model::ListCollectionsOutcome outcome =
                rekognitionClient.ListCollections(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String>& collectionsIds =
 outcome.GetResult().GetCollectionIds();
            if (!collectionsIds.empty()) {
                std::cout << "collectionsIds: " << std::endl;
                for (auto &collectionId : collectionsIds) {
                    std::cout << "- " << collectionId << std::endl;
                }
            } else {
                std::cout << "No collections found" << std::endl;
            }
        } else {
            std::cerr << "Error with ListCollections: " << outcome.GetError()
                      << std::endl;
        }
```

```
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考ListCollections中的。

程式碼範例

- 使用 SDK 執行 Amazon Rekognition 的動作 AWS
  - 搭CompareFaces配 AWS 開發套件或 CLI 使用
  - 搭CreateCollection配 AWS 開發套件或 CLI 使用
  - 搭DeleteCollection配 AWS 開發套件或 CLI 使用
  - 搭DeleteFaces配 AWS 開發套件或 CLI 使用
  - 搭DescribeCollection配 AWS 開發套件或 CLI 使用
  - 搭DetectFaces配 AWS 開發套件或 CLI 使用
  - 搭DetectLabels配 AWS 開發套件或 CLI 使用
  - 搭DetectModerationLabels配 AWS 開發套件或 CLI 使用
  - 搭DetectText配 AWS 開發套件或 CLI 使用
  - 搭DisassociateFaces配 AWS 開發套件或 CLI 使用
  - 搭GetCelebrityInfo配 AWS 開發套件或 CLI 使用
  - 搭IndexFaces配 AWS 開發套件或 CLI 使用
  - 搭ListCollections配 AWS 開發套件或 CLI 使用
  - 搭ListFaces配 AWS 開發套件或 CLI 使用
  - 搭RecognizeCelebrities配 AWS 開發套件或 CLI 使用
  - 搭SearchFaces配 AWS 開發套件或 CLI 使用
  - 搭SearchFacesByImage配 AWS 開發套件或 CLI 使用
- 使用 SDK 的 Amazon Rekognition 案例 AWS
  - 建立 Amazon Rekognition 集合，並使用開發套件在其中尋找臉孔 AWS
  - 使用開發套件使用 Amazon Rekognition 偵測和顯示映像中的元素 AWS
  - 使用 Amazon Rekognition 和開發套件偵測影片中的資訊 AWS

- 使用 SDK 的 Amazon Rekognition 跨服務範例 AWS
  - 建立相片資產管理應用程式，讓使用者以標籤管理相片
  - 使用開發套件使用 Amazon 重新認知功能偵測影像中的個人防護裝置 AWS
  - 使用 AWS SDK 偵測影像中的臉孔
  - 使用開發套件使用 Amazon Rekognition 偵測影像中的物件 AWS
  - 使用開發套件使用 Amazon Rekognition 偵測影片中的人物和物件 AWS
  - 使用 SDK 儲存 EXIF 和其他影像資訊 AWS

使用 SDK 執行 Amazon Rekognition 的動作 AWS

下列程式碼範例示範如何使用開發套件執行個別 Amazon Rekognition 動作。 AWS 這些摘錄會呼叫 Amazon Rekognition API，是必須在內容中執行的大型程式的程式碼摘錄。每個範例都包含一個連結 GitHub，您可以在其中找到設定和執行程式碼的指示。

下列範例僅包含最常使用的動作。如需完整清單，請參閱 Amazon Rekognition API 參考。

範例

- 搭CompareFaces配 AWS 開發套件或 CLI 使用
- 搭CreateCollection配 AWS 開發套件或 CLI 使用
- 搭DeleteCollection配 AWS 開發套件或 CLI 使用
- 搭DeleteFaces配 AWS 開發套件或 CLI 使用
- 搭DescribeCollection配 AWS 開發套件或 CLI 使用
- 搭DetectFaces配 AWS 開發套件或 CLI 使用
- 搭DetectLabels配 AWS 開發套件或 CLI 使用
- 搭DetectModerationLabels配 AWS 開發套件或 CLI 使用
- 搭DetectText配 AWS 開發套件或 CLI 使用
- 搭DisassociateFaces配 AWS 開發套件或 CLI 使用
- 搭GetCelebrityInfo配 AWS 開發套件或 CLI 使用
- 搭IndexFaces配 AWS 開發套件或 CLI 使用
- 搭ListCollections配 AWS 開發套件或 CLI 使用
- 搭ListFaces配 AWS 開發套件或 CLI 使用
- 搭RecognizeCelebrities配 AWS 開發套件或 CLI 使用
- 搭SearchFaces配 AWS 開發套件或 CLI 使用

- 搭SearchFacesByImage配 AWS 開發套件或 CLI 使用

搭CompareFaces配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用CompareFaces。

如需詳細資訊，請參閱比較映像中的人臉。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to compare faces in two images.
    /// </summary>
    public class CompareFaces
    {
        public static async Task Main()
        {
            float similarityThreshold = 70F;
            string sourceImage = "source.jpg";
            string targetImage = "target.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            Amazon.Rekognition.Model.Image imageSource = new
Amazon.Rekognition.Model.Image();

            try
            {
```

```
                using FileStream fs = new FileStream(sourceImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageSource.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine($"Failed to load source image: {sourceImage}");
                return;
            }

            Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();

            try
            {
                using FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageTarget.Bytes = new MemoryStream(data);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Failed to load target image: {targetImage}");
                Console.WriteLine(ex.Message);
                return;
            }

            var compareFacesRequest = new CompareFacesRequest
            {
                SourceImage = imageSource,
                TargetImage = imageTarget,
                SimilarityThreshold = similarityThreshold,
            };

            // Call operation
            var compareFacesResponse = await
rekognitionClient.CompareFacesAsync(compareFacesRequest);

            // Display results
            compareFacesResponse.FaceMatches.ForEach(match =>
```

```
            {
                ComparedFace face = match.Face;
                BoundingBox position = face.BoundingBox;
                Console.WriteLine($"Face at {position.Left} {position.Top}
 matches with {match.Similarity}% confidence.");
            });

            Console.WriteLine($"Found {compareFacesResponse.UnmatchedFaces.Count}
 face(s) that did not match.");
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考CompareFaces中的。

CLI

AWS CLI

比較兩個影像中的臉孔

以下compare-faces命令比較存放在 Amazon S3 儲存貯體中的兩個映像中的臉孔。

```
aws rekognition compare-faces \
    --source-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"source.jpg"}}' \
    --target-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"target.jpg"}}'
```

輸出：

```
{
    "UnmatchedFaces": [],
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.12368916720151901,
                    "Top": 0.16007372736930847,
                    "Left": 0.5901257991790771,
                    "Height": 0.25140416622161865
                },
```

```
                "Confidence": 100.0,
                "Pose": {
                    "Yaw": -3.7351467609405518,
                    "Roll": -0.10309021919965744,
                    "Pitch": 0.8637830018997192
                },
                "Quality": {
                    "Sharpness": 95.51618957519531,
                    "Brightness": 65.29893493652344
                },
                "Landmarks": [
                    {
                        "Y": 0.26721030473709106,
                        "X": 0.6204193830490112,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.26831310987472534,
                        "X": 0.6776827573776245,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.3514654338359833,
                        "X": 0.6241428852081299,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.35258132219314575,
                        "X": 0.6713621020317078,
                        "Type": "mouthRight"
                    },
                    {
                        "Y": 0.3140771687030792,
                        "X": 0.6428444981575012,
                        "Type": "nose"
                    }
                ]
            },
            "Similarity": 100.0
        }
    ],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.12368916720151901,
```

```
            "Top": 0.16007372736930847,
            "Left": 0.5901257991790771,
            "Height": 0.25140416622161865
        },
        "Confidence": 100.0
    }
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的比較映像中的人臉。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CompareFaces中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.CompareFacesRequest;
import software.amazon.awssdk.services.rekognition.model.CompareFacesResponse;
import software.amazon.awssdk.services.rekognition.model.CompareFacesMatch;
import software.amazon.awssdk.services.rekognition.model.ComparedFace;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CompareFaces {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <pathSource> <pathTarget>

                Where:
                   pathSource - The path to the source image (for example, C:\
\AWS\\pic1.png).\s
                    pathTarget - The path to the target image (for example, C:\
\AWS\\pic2.png).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        Float similarityThreshold = 70F;
        String sourceImage = args[0];
        String targetImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        compareTwoFaces(rekClient, similarityThreshold, sourceImage,
 targetImage);
        rekClient.close();
    }

    public static void compareTwoFaces(RekognitionClient rekClient, Float
 similarityThreshold, String sourceImage,
            String targetImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            InputStream tarStream = new FileInputStream(targetImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            SdkBytes targetBytes = SdkBytes.fromInputStream(tarStream);
```

```
            // Create an Image object for the source image.
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            Image tarImage = Image.builder()
                    .bytes(targetBytes)
                    .build();

            CompareFacesRequest facesRequest = CompareFacesRequest.builder()
                    .sourceImage(souImage)
                    .targetImage(tarImage)
                    .similarityThreshold(similarityThreshold)
                    .build();

            // Compare the two images.
            CompareFacesResponse compareFacesResult =
 rekClient.compareFaces(facesRequest);
            List<CompareFacesMatch> faceDetails =
 compareFacesResult.faceMatches();
            for (CompareFacesMatch match : faceDetails) {
                ComparedFace face = match.face();
                BoundingBox position = face.boundingBox();
                System.out.println("Face at " + position.left().toString()
                        + " " + position.top()
                        + " matches with " + face.confidence().toString()
                        + "% confidence.");

            }
            List<ComparedFace> uncompared = compareFacesResult.unmatchedFaces();
            System.out.println("There was " + uncompared.size() + " face(s) that
 did not match");
            System.out.println("Source image rotation: " +
 compareFacesResult.sourceImageOrientationCorrection());
            System.out.println("target image rotation: " +
 compareFacesResult.targetImageOrientationCorrection());

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println("Failed to load source image " + sourceImage);
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考CompareFaces中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun compareTwoFaces(
    similarityThresholdVal: Float,
    sourceImageVal: String,
    targetImageVal: String,
) {
    val sourceBytes = (File(sourceImageVal).readBytes())
    val targetBytes = (File(targetImageVal).readBytes())

    // Create an Image object for the source image.
    val souImage =
        Image {
            bytes = sourceBytes
        }

    val tarImage =
        Image {
            bytes = targetBytes
        }

    val facesRequest =
        CompareFacesRequest {
            sourceImage = souImage
            targetImage = tarImage
            similarityThreshold = similarityThresholdVal
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
```

```
        val compareFacesResult = rekClient.compareFaces(facesRequest)
        val faceDetails = compareFacesResult.faceMatches

        if (faceDetails != null) {
            for (match: CompareFacesMatch in faceDetails) {
                val face = match.face
                val position = face?.boundingBox
                if (position != null) {
                    println("Face at ${position.left} ${position.top} matches
 with ${face.confidence} % confidence.")
                }
            }
        }

        val uncompared = compareFacesResult.unmatchedFaces
        if (uncompared != null) {
            println("There was ${uncompared.size} face(s) that did not match")
        }

        println("Source image rotation:
 ${compareFacesResult.sourceImageOrientationCorrection}")
        println("target image rotation:
 ${compareFacesResult.targetImageOrientationCorrection}")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK CompareFaces中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
```

```
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def compare_faces(self, target_image, similarity):
        """
        Compares faces in the image with the largest face in the target image.

        :param target_image: The target image to compare against.
        :param similarity: Faces in the image must have a similarity value
greater
                           than this value to be included in the results.
        :return: A tuple. The first element is the list of faces that match the
                 reference image. The second element is the list of faces that
have
                 a similarity value below the specified threshold.
        """
        try:
            response = self.rekognition_client.compare_faces(
                SourceImage=self.image,
                TargetImage=target_image.image,
                SimilarityThreshold=similarity,
            )
            matches = [
                RekognitionFace(match["Face"]) for match in
response["FaceMatches"]
            ]
            unmatches = [RekognitionFace(face) for face in
response["UnmatchedFaces"]]
            logger.info(
                "Found %s matched faces and %s unmatched faces.",
```

```
                len(matches),
                len(unmatches),
            )
        except ClientError:
            logger.exception(
                "Couldn't match faces from %s to %s.",
                self.image_name,
                target_image.image_name,
            )
            raise
        else:
            return matches, unmatches
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件CompareFaces中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭CreateCollection配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用CreateCollection。

如需更多資訊，請參閱建立集合。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
    /// <summary>
    /// Uses Amazon Rekognition to create a collection to which you can add
    /// faces using the IndexFaces operation.
    /// </summary>
    public class CreateCollection
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine("Creating collection: " + collectionId);

            var createCollectionRequest = new CreateCollectionRequest
            {
                CollectionId = collectionId,
            };

            CreateCollectionResponse createCollectionResponse = await
rekognitionClient.CreateCollectionAsync(createCollectionRequest);
            Console.WriteLine($"CollectionArn :
{createCollectionResponse.CollectionArn}");
            Console.WriteLine($"Status code :
{createCollectionResponse.StatusCode}");
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考CreateCollection中的。

CLI

AWS CLI

建立集合的步驟

下面的`create-collection`命令創建具有指定名稱的集合。

```
aws rekognition create-collection \
    --collection-id "MyCollection"
```

輸出：

```
{
    "CollectionArn": "aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0",
    "StatusCode": 200
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的建立集合。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CreateCollection中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.CreateCollectionRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateCollection {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:    <collectionName>\s

                Where:
                   collectionName - The name of the collection.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Creating collection: " + collectionId);
        createMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    public static void createMyCollection(RekognitionClient rekClient, String
 collectionId) {
        try {
            CreateCollectionRequest collectionRequest =
 CreateCollectionRequest.builder()
                    .collectionId(collectionId)
                    .build();

            CreateCollectionResponse collectionResponse =
 rekClient.createCollection(collectionRequest);
            System.out.println("CollectionArn: " +
 collectionResponse.collectionArn());
            System.out.println("Status code: " +
 collectionResponse.statusCode().toString());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考CreateCollection中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun createMyCollection(collectionIdVal: String) {
    val request =
        CreateCollectionRequest {
            collectionId = collectionIdVal
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.createCollection(request)
        println("Collection ARN is ${response.collectionArn}")
        println("Status code is ${response.statusCode}")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK CreateCollection中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
 API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client


    def create_collection(self, collection_id):
        """
        Creates an empty collection.

        :param collection_id: Text that identifies the collection.
        :return: The newly created collection.
        """
        try:
            response = self.rekognition_client.create_collection(
                CollectionId=collection_id
            )
            response["CollectionId"] = collection_id
            collection = RekognitionCollection(response, self.rekognition_client)
            logger.info("Created collection %s.", collection_id)
        except ClientError:
            logger.exception("Couldn't create collection %s.", collection_id)
            raise
        else:
            return collection
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件CreateCollection中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DeleteCollection配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DeleteCollection。

如需更多資訊，請參閱刪除集合。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to delete an existing collection.
/// </summary>
public class DeleteCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine("Deleting collection: " + collectionId);

        var deleteCollectionRequest = new DeleteCollectionRequest()
        {
            CollectionId = collectionId,
        };
```

```
            var deleteCollectionResponse = await
rekognitionClient.DeleteCollectionAsync(deleteCollectionRequest);
            Console.WriteLine($"{collectionId}:
{deleteCollectionResponse.StatusCode}");
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DeleteCollection中的。

CLI

AWS CLI

若要刪除商品系列

下面的delete-collection命令刪除指定的集合。

```
aws rekognition delete-collection \
    --collection-id MyCollection
```

輸出：

```
{
    "StatusCode": 200
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的刪除集合。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DeleteCollection中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId>\s

                Where:
                   collectionId - The id of the collection to delete.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteMyCollection(rekClient, collectionId);
        rekClient.close();
    }
```

```
    public static void deleteMyCollection(RekognitionClient rekClient, String
 collectionId) {
        try {
            DeleteCollectionRequest deleteCollectionRequest =
 DeleteCollectionRequest.builder()
                    .collectionId(collectionId)
                    .build();

            DeleteCollectionResponse deleteCollectionResponse =
 rekClient.deleteCollection(deleteCollectionRequest);
            System.out.println(collectionId + ": " +
 deleteCollectionResponse.statusCode().toString());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DeleteCollection中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun deleteMyCollection(collectionIdVal: String) {
    val request =
        DeleteCollectionRequest {
            collectionId = collectionIdVal
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.deleteCollection(request)
        println("The collectionId status is ${response.statusCode}")
```

```
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK DeleteCollection中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.
```

```
        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def delete_collection(self):
        """
        Deletes the collection.
        """
        try:

self.rekognition_client.delete_collection(CollectionId=self.collection_id)
            logger.info("Deleted collection %s.", self.collection_id)
            self.collection_id = None
        except ClientError:
            logger.exception("Couldn't delete collection %s.",
self.collection_id)
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteCollection中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DeleteFaces配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DeleteFaces。

如需詳細資訊，請參閱從集合中刪除人臉。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete one or more faces from
    /// a Rekognition collection.
    /// </summary>
    public class DeleteFaces
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" };

            var rekognitionClient = new AmazonRekognitionClient();

            var deleteFacesRequest = new DeleteFacesRequest()
            {
                CollectionId = collectionId,
                FaceIds = faces,
            };

            DeleteFacesResponse deleteFacesResponse = await
 rekognitionClient.DeleteFacesAsync(deleteFacesRequest);
            deleteFacesResponse.DeletedFaces.ForEach(face =>
            {
                Console.WriteLine($"FaceID: {face}");
            });
```

```
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DeleteFaces中的。

CLI

AWS CLI

從集合中刪除面的步驟

以下delete-faces命令從集合中刪除指定的面。

```
aws rekognition delete-faces \
    --collection-id MyCollection
    --face-ids '["0040279c-0178-436e-b70a-e61b074e96b0"]'
```

輸出：

```
{
    "DeletedFaces": [
        "0040279c-0178-436e-b70a-e61b074e96b0"
    ]
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的刪除集合中的臉孔。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DeleteFaces中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteFacesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFacesFromCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <faceId>\s

                Where:
                   collectionId - The id of the collection from which faces are
 deleted.\s

                   faceId - The id of the face to delete.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteFacesCollection(rekClient, collectionId, faceId);
        rekClient.close();
    }
```

```
    public static void deleteFacesCollection(RekognitionClient rekClient,
            String collectionId,
            String faceId) {

        try {
            DeleteFacesRequest deleteFacesRequest = DeleteFacesRequest.builder()
                    .collectionId(collectionId)
                    .faceIds(faceId)
                    .build();

            rekClient.deleteFaces(deleteFacesRequest);
            System.out.println("The face was deleted from the collection.");

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DeleteFaces中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun deleteFacesCollection(
    collectionIdVal: String?,
    faceIdVal: String,
) {
    val deleteFacesRequest =
        DeleteFacesRequest {
            collectionId = collectionIdVal
            faceIds = listOf(faceIdVal)
```

```
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        rekClient.deleteFaces(deleteFacesRequest)
        println("$faceIdVal was deleted from the collection")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK DeleteFaces中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client
```

```
    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def delete_faces(self, face_ids):
        """
        Deletes faces from the collection.

        :param face_ids: The list of IDs of faces to delete.
        :return: The list of IDs of faces that were deleted.
        """
        try:
            response = self.rekognition_client.delete_faces(
                CollectionId=self.collection_id, FaceIds=face_ids
            )
            deleted_ids = response["DeletedFaces"]
            logger.info(
                "Deleted %s faces from %s.", len(deleted_ids), self.collection_id
            )
        except ClientError:
            logger.exception("Couldn't delete faces from %s.",
self.collection_id)
            raise
        else:
            return deleted_ids
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteFaces中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DescribeCollection配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DescribeCollection。

如需詳細資訊，請參閱描述集合。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to describe the contents of a
/// collection.
/// </summary>
public class DescribeCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine($"Describing collection: {collectionId}");

        var describeCollectionRequest = new DescribeCollectionRequest()
        {
            CollectionId = collectionId,
        };
```

```
            var describeCollectionResponse = await
 rekognitionClient.DescribeCollectionAsync(describeCollectionRequest);
            Console.WriteLine($"Collection ARN:
 {describeCollectionResponse.CollectionARN}");
            Console.WriteLine($"Face count:
 {describeCollectionResponse.FaceCount}");
            Console.WriteLine($"Face model version:
 {describeCollectionResponse.FaceModelVersion}");
            Console.WriteLine($"Created:
 {describeCollectionResponse.CreationTimestamp}");
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DescribeCollection中的。

CLI

AWS CLI

描述系列

下列describe-collection範例會顯示有關指定集合的詳細資訊。

```
aws rekognition describe-collection \
    --collection-id MyCollection
```

輸出：

```
{
    "FaceCount": 200,
    "CreationTimestamp": 1569444828.274,
    "CollectionARN": "arn:aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0"
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的說明集合。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DescribeCollection中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionName>

                Where:
                   collectionName - The name of the Amazon Rekognition
 collection.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String collectionName = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        describeColl(rekClient, collectionName);
        rekClient.close();
    }

    public static void describeColl(RekognitionClient rekClient, String
 collectionName) {
        try {
            DescribeCollectionRequest describeCollectionRequest =
 DescribeCollectionRequest.builder()
                    .collectionId(collectionName)
                    .build();

            DescribeCollectionResponse describeCollectionResponse = rekClient
                    .describeCollection(describeCollectionRequest);
            System.out.println("Collection Arn : " +
 describeCollectionResponse.collectionARN());
            System.out.println("Created : " +
 describeCollectionResponse.creationTimestamp().toString());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DescribeCollection中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun describeColl(collectionName: String) {
    val request =
        DescribeCollectionRequest {
            collectionId = collectionName
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.describeCollection(request)
        println("The collection Arn is ${response.collectionArn}")
        println("The collection contains this many faces ${response.faceCount}")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK DescribeCollection中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
```

```
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def describe_collection(self):
        """
        Gets data about the collection from the Amazon Rekognition service.

        :return: The collection rendered as a dict.
        """
        try:
            response = self.rekognition_client.describe_collection(
                CollectionId=self.collection_id
            )
            # Work around capitalization of Arn vs. ARN
```

```
            response["CollectionArn"] = response.get("CollectionARN")
            (
                self.collection_arn,
                self.face_count,
                self.created,
            ) = self._unpack_collection(response)
            logger.info("Got data for collection %s.", self.collection_id)
        except ClientError:
            logger.exception("Couldn't get data for collection %s.",
self.collection_id)
            raise
        else:
            return self.to_dict()
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DescribeCollection中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DetectFaces配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DetectFaces。

如需詳細資訊，請參閱在映像中偵測人臉。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect faces within an image
    /// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
    /// </summary>
    public class DetectFaces
    {
        public static async Task Main()
        {
            string photo = "input.jpg";
            string bucket = "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectFacesRequest = new DetectFacesRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },

                // Attributes can be "ALL" or "DEFAULT".
                // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and
 Quality.
                // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/
items/Rekognition/TFaceDetail.html
                Attributes = new List<string>() { "ALL" },
            };

            try
            {
                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
                foreach (FaceDetail face in detectFacesResponse.FaceDetails)
                {
```

```
                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
left={face.BoundingBox.Top} width={face.BoundingBox.Width}
height={face.BoundingBox.Height}");
                    Console.WriteLine($"Confidence: {face.Confidence}");
                    Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
                    Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
                    Console.WriteLine($"Brightness:
{face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

                    if (hasAll)
                    {
                        Console.WriteLine($"Estimated age is between
{face.AgeRange.Low} and {face.AgeRange.High} years old.");
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

顯示映像中所有人臉的邊界框資訊。

```
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to display the details of the
    /// bounding boxes around the faces detected in an image.
    /// </summary>
    public class ImageOrientationBoundingBox
    {
        public static async Task Main()
```

```
        {
            string photo = @"D:\Development\AWS-Examples\Rekognition
\target.jpg"; // "photo.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            int height;
            int width;

            // Used to extract original photo width/height
            using (var imageBitmap = new Bitmap(photo))
            {
                height = imageBitmap.Height;
                width = imageBitmap.Width;
            }

            Console.WriteLine("Image Information:");
            Console.WriteLine(photo);
            Console.WriteLine("Image Height: " + height);
            Console.WriteLine("Image Width: " + width);

            try
            {
                var detectFacesRequest = new DetectFacesRequest()
                {
                    Image = image,
                    Attributes = new List<string>() { "ALL" },
                };
```

```
                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                detectFacesResponse.FaceDetails.ForEach(face =>
                {
                    Console.WriteLine("Face:");
                    ShowBoundingBoxPositions(
                        height,
                        width,
                        face.BoundingBox,
                        detectFacesResponse.OrientationCorrection);

                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"The detected face is estimated to be
 between {face.AgeRange.Low} and {face.AgeRange.High} years old.\n");
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Display the bounding box information for an image.
        /// </summary>
        /// <param name="imageHeight">The height of the image.</param>
        /// <param name="imageWidth">The width of the image.</param>
        /// <param name="box">The bounding box for a face found within the
 image.</param>
        /// <param name="rotation">The rotation of the face's bounding box.</
param>
        public static void ShowBoundingBoxPositions(int imageHeight, int
 imageWidth, BoundingBox box, string rotation)
        {
            float left;
            float top;

            if (rotation == null)
            {
                Console.WriteLine("No estimated orientation. Check Exif data.");
                return;
```

```
            }

            // Calculate face position based on image orientation.
            switch (rotation)
            {
                case "ROTATE_0":
                    left = imageWidth * box.Left;
                    top = imageHeight * box.Top;
                    break;
                case "ROTATE_90":
                    left = imageHeight * (1 - (box.Top + box.Height));
                    top = imageWidth * box.Left;
                    break;
                case "ROTATE_180":
                    left = imageWidth - (imageWidth * (box.Left + box.Width));
                    top = imageHeight * (1 - (box.Top + box.Height));
                    break;
                case "ROTATE_270":
                    left = imageHeight * box.Top;
                    top = imageWidth * (1 - box.Left - box.Width);
                    break;
                default:
                    Console.WriteLine("No estimated orientation information.
Check Exif data.");
                    return;
            }

            // Display face location information.
            Console.WriteLine($"Left: {left}");
            Console.WriteLine($"Top: {top}");
            Console.WriteLine($"Face Width: {imageWidth * box.Width}");
            Console.WriteLine($"Face Height: {imageHeight * box.Height}");
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DetectFaces中的。

CLI

AWS CLI

偵測影像中的臉部

下列`detect-faces`命令會偵測存放在 Amazon S3 儲存貯體中的指定映像中的臉孔。

```
aws rekognition detect-faces \
    --image '{"S3Object":{"Bucket":"MyImageS3Bucket","Name":"MyFriend.jpg"}}' \
    --attributes "ALL"
```

輸出：

```
{
    "FaceDetails": [
        {
            "Confidence": 100.0,
            "Eyeglasses": {
                "Confidence": 98.91107940673828,
                "Value": false
            },
            "Sunglasses": {
                "Confidence": 99.7966537475586,
                "Value": false
            },
            "Gender": {
                "Confidence": 99.56611633300781,
                "Value": "Male"
            },
            "Landmarks": [
                {
                    "Y": 0.26721030473709106,
                    "X": 0.6204193830490112,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.26831310987472534,
                    "X": 0.6776827573776245,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3514654338359833,
```

```
                    "X": 0.6241428852081299,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.35258132219314575,
                    "X": 0.6713621020317078,
                    "Type": "mouthRight"
                },
                {
                    "Y": 0.3140771687030792,
                    "X": 0.6428444981575012,
                    "Type": "nose"
                },
                {
                    "Y": 0.24662546813488007,
                    "X": 0.6001564860343933,
                    "Type": "leftEyeBrowLeft"
                },
                {
                    "Y": 0.24326619505882263,
                    "X": 0.6303644776344299,
                    "Type": "leftEyeBrowRight"
                },
                {
                    "Y": 0.23818562924861908,
                    "X": 0.6146903038024902,
                    "Type": "leftEyeBrowUp"
                },
                {
                    "Y": 0.24373626708984375,
                    "X": 0.6640064716339111,
                    "Type": "rightEyeBrowLeft"
                },
                {
                    "Y": 0.24877218902111053,
                    "X": 0.7025929093360901,
                    "Type": "rightEyeBrowRight"
                },
                {
                    "Y": 0.23938551545143127,
                    "X": 0.6823262572288513,
                    "Type": "rightEyeBrowUp"
                },
                {
```

```
                    "Y": 0.265746533870697,
                    "X": 0.6112898588180542,
                    "Type": "leftEyeLeft"
                },
                {
                    "Y": 0.2676128149032593,
                    "X": 0.6317071914672852,
                    "Type": "leftEyeRight"
                },
                {
                    "Y": 0.262735515832901,
                    "X": 0.6201658248901367,
                    "Type": "leftEyeUp"
                },
                {
                    "Y": 0.27025148272514343,
                    "X": 0.6206279993057251,
                    "Type": "leftEyeDown"
                },
                {
                    "Y": 0.268223375082016,
                    "X": 0.6658390760421753,
                    "Type": "rightEyeLeft"
                },
                {
                    "Y": 0.2672517001628876,
                    "X": 0.687832236289978,
                    "Type": "rightEyeRight"
                },
                {
                    "Y": 0.26383838057518005,
                    "X": 0.6769183874130249,
                    "Type": "rightEyeUp"
                },
                {
                    "Y": 0.27138751745224,
                    "X": 0.676596462726593,
                    "Type": "rightEyeDown"
                },
                {
                    "Y": 0.32283174991607666,
                    "X": 0.6350004076957703,
                    "Type": "noseLeft"
                },
```

```
                {
                    "Y": 0.3219289481639862,
                    "X": 0.6567046642303467,
                    "Type": "noseRight"
                },
                {
                    "Y": 0.3420318365097046,
                    "X": 0.6450609564781189,
                    "Type": "mouthUp"
                },
                {
                    "Y": 0.3664324879646301,
                    "X": 0.6455618143081665,
                    "Type": "mouthDown"
                },
                {
                    "Y": 0.26721030473709106,
                    "X": 0.6204193830490112,
                    "Type": "leftPupil"
                },
                {
                    "Y": 0.26831310987472534,
                    "X": 0.6776827573776245,
                    "Type": "rightPupil"
                },
                {
                    "Y": 0.26343393325805664,
                    "X": 0.5946047306060791,
                    "Type": "upperJawlineLeft"
                },
                {
                    "Y": 0.3543180525302887,
                    "X": 0.6044883728027344,
                    "Type": "midJawlineLeft"
                },
                {
                    "Y": 0.4084877669811249,
                    "X": 0.6477024555206299,
                    "Type": "chinBottom"
                },
                {
                    "Y": 0.3562754988670349,
                    "X": 0.707981526851654,
                    "Type": "midJawlineRight"
```

```
                },
                {
                    "Y": 0.26580461859703064,
                    "X": 0.7234612107276917,
                    "Type": "upperJawlineRight"
                }
            ],
            "Pose": {
                "Yaw": -3.7351467609405518,
                "Roll": -0.10309021919965744,
                "Pitch": 0.8637830018997192
            },
            "Emotions": [
                {
                    "Confidence": 8.74203109741211,
                    "Type": "SURPRISED"
                },
                {
                    "Confidence": 2.501944065093994,
                    "Type": "ANGRY"
                },
                {
                    "Confidence": 0.7378743290901184,
                    "Type": "DISGUSTED"
                },
                {
                    "Confidence": 3.5296201705932617,
                    "Type": "HAPPY"
                },
                {
                    "Confidence": 1.7162904739379883,
                    "Type": "SAD"
                },
                {
                    "Confidence": 9.518536567687988,
                    "Type": "CONFUSED"
                },
                {
                    "Confidence": 0.45474427938461304,
                    "Type": "FEAR"
                },
                {
                    "Confidence": 72.79895782470703,
                    "Type": "CALM"
```

```
                }
            ],
            "AgeRange": {
                "High": 48,
                "Low": 32
            },
            "EyesOpen": {
                "Confidence": 98.93987274169922,
                "Value": true
            },
            "BoundingBox": {
                "Width": 0.12368916720151901,
                "Top": 0.16007372736930847,
                "Left": 0.5901257991790771,
                "Height": 0.25140416622161865
            },
            "Smile": {
                "Confidence": 93.4493179321289,
                "Value": false
            },
            "MouthOpen": {
                "Confidence": 90.53053283691406,
                "Value": false
            },
            "Quality": {
                "Sharpness": 95.51618957519531,
                "Brightness": 65.29893493652344
            },
            "Mustache": {
                "Confidence": 89.85221099853516,
                "Value": false
            },
            "Beard": {
                "Confidence": 86.1991195678711,
                "Value": true
            }
        }
    ]
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的偵測影像中的人臉。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DetectFaces中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.AgeRange;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectFaces {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <sourceImage>

                Where:
```

```
                    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectFacesinImage(rekClient, sourceImage);
        rekClient.close();
    }

    public static void detectFacesinImage(RekognitionClient rekClient, String
 sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            DetectFacesRequest facesRequest = DetectFacesRequest.builder()
                    .attributes(Attribute.ALL)
                    .image(souImage)
                    .build();

            DetectFacesResponse facesResponse =
 rekClient.detectFaces(facesRequest);
            List<FaceDetail> faceDetails = facesResponse.faceDetails();
            for (FaceDetail face : faceDetails) {
                AgeRange ageRange = face.ageRange();
                System.out.println("The detected face is estimated to be between
 "
                        + ageRange.low().toString() + " and " +
 ageRange.high().toString()
```

```
                        + " years old.");

                System.out.println("There is a smile : " +
 face.smile().value().toString());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DetectFaces中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun detectFacesinImage(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectFacesRequest {
            attributes = listOf(Attribute.All)
            image = souImage
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectFaces(request)
        response.faceDetails?.forEach { face ->
            val ageRange = face.ageRange
```

```
            println("The detected face is estimated to be between
 ${ageRange?.low} and ${ageRange?.high} years old.")
            println("There is a smile ${face.smile?.value}")
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK DetectFaces中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_faces(self):
        """
```

```
        Detects faces in the image.

        :return: The list of faces found in the image.
        """
        try:
            response = self.rekognition_client.detect_faces(
                Image=self.image, Attributes=["ALL"]
            )
            faces = [RekognitionFace(face) for face in response["FaceDetails"]]
            logger.info("Detected %s faces.", len(faces))
        except ClientError:
            logger.exception("Couldn't detect faces in %s.", self.image_name)
            raise
        else:
            return faces
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DetectFaces中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DetectLabels配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DetectLabels。

如需詳細資訊，請參閱偵測映像中的標籤。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
    /// </summary>
    public class DetectLabels
    {
        public static async Task Main()
        {
            string photo = "del_river_02092020_01.jpg"; // "input.jpg";
            string bucket = "igsmiths3photos"; // "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
                MaxLabels = 10,
                MinConfidence = 75F,
            };

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
```

```
            }
        }
    }
```

偵測儲存於您計算機的映像檔案中的標籤。

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored locally.
    /// </summary>
    public class DetectLabelsLocalFile
    {
        public static async Task Main()
        {
            string photo = "input.jpg";

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
```

```
            {
                Image = image,
                MaxLabels = 10,
                MinConfidence = 77F,
            };

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
 rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine($"Detected labels for {photo}");
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"{label.Name}: {label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DetectLabels中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
//! Detect instances of real-world entities within an image by using Amazon
 Rekognition
/*!
  \param imageBucket: The Amazon Simple Storage Service (Amazon S3) bucket
 containing an image.
```

```
  \param imageKey: The Amazon S3 key of an image object.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::Rekognition::detectLabels(const Aws::String &imageBucket,
                                       const Aws::String &imageKey,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::Rekognition::RekognitionClient rekognitionClient(clientConfiguration);

    Aws::Rekognition::Model::DetectLabelsRequest request;
    Aws::Rekognition::Model::S3Object s3Object;
    s3Object.SetBucket(imageBucket);
    s3Object.SetName(imageKey);

    Aws::Rekognition::Model::Image image;
    image.SetS3Object(s3Object);

    request.SetImage(image);

    const Aws::Rekognition::Model::DetectLabelsOutcome outcome =
 rekognitionClient.DetectLabels(request);

    if (outcome.IsSuccess()) {
        const Aws::Vector<Aws::Rekognition::Model::Label> &labels =
 outcome.GetResult().GetLabels();
        if (labels.empty()) {
            std::cout << "No labels detected" << std::endl;
        } else {
            for (const Aws::Rekognition::Model::Label &label: labels) {
                std::cout << label.GetName() << ": " << label.GetConfidence() <<
 std::endl;
            }
        }
    } else {
        std::cerr << "Error while detecting labels: '"
                  << outcome.GetError().GetMessage()
                  << "'" << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考DetectLabels中的。

CLI

AWS CLI

偵測影像中的標籤

下列detect-labels範例會偵測存放在 Amazon S3 儲存貯體中的映像中的場景和物件。

```
aws rekognition detect-labels \
    --image '{"S3Object":{"Bucket":"bucket","Name":"image"}}'
```

輸出：

```
{
    "Labels": [
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Automobile"
        },
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Vehicle"
        },
        {
```

```
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [],
            "Name": "Transportation"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.10616336017847061,
                        "Top": 0.5039216876029968,
                        "Left": 0.0037978808395564556,
                        "Height": 0.18528179824352264
                    },
                    "Confidence": 99.15271759033203
                },
                {
                    "BoundingBox": {
                        "Width": 0.2429988533258438,
                        "Top": 0.5251884460449219,
                        "Left": 0.7309805154800415,
                        "Height": 0.21577216684818268
                    },
                    "Confidence": 99.1286392211914
                },
                {
                    "BoundingBox": {
                        "Width": 0.14233611524105072,
                        "Top": 0.5333095788955688,
                        "Left": 0.6494812965393066,
                        "Height": 0.15528248250484467
                    },
                    "Confidence": 98.48368072509766
                },
                {
                    "BoundingBox": {
                        "Width": 0.11086395382881165,
                        "Top": 0.5354844927787781,
                        "Left": 0.10355594009160995,
                        "Height": 0.10271988064050674
                    },
                    "Confidence": 96.45606231689453
                },
                {
```

```
                    "BoundingBox": {
                        "Width": 0.06254628300666809,
                        "Top": 0.5573825240135193,
                        "Left": 0.46083059906959534,
                        "Height": 0.053911514580249786
                    },
                    "Confidence": 93.65448760986328
                },
                {
                    "BoundingBox": {
                        "Width": 0.10105438530445099,
                        "Top": 0.534368634223938,
                        "Left": 0.5743985772132874,
                        "Height": 0.12226245552301407
                    },
                    "Confidence": 93.06217193603516
                },
                {
                    "BoundingBox": {
                        "Width": 0.056389667093753815,
                        "Top": 0.5235804319381714,
                        "Left": 0.9427769780158997,
                        "Height": 0.17163699865341187
                    },
                    "Confidence": 92.6864013671875
                },
                {
                    "BoundingBox": {
                        "Width": 0.06003860384225845,
                        "Top": 0.5441341400146484,
                        "Left": 0.22409997880458832,
                        "Height": 0.06737709045410156
                    },
                    "Confidence": 90.4227066040039
                },
                {
                    "BoundingBox": {
                        "Width": 0.02848697081208229,
                        "Top": 0.5107086896896362,
                        "Left": 0,
                        "Height": 0.19150497019290924
                    },
                    "Confidence": 86.65286254882812
                },
```

```
                {
                    "BoundingBox": {
                        "Width": 0.04067881405353546,
                        "Top": 0.5566273927688599,
                        "Left": 0.316415935754776,
                        "Height": 0.03428703173995018
                    },
                    "Confidence": 85.36471557617188
                },
                {
                    "BoundingBox": {
                        "Width": 0.043411049991846085,
                        "Top": 0.5394920110702515,
                        "Left": 0.18293385207653046,
                        "Height": 0.0893595889210701
                    },
                    "Confidence": 82.21705627441406
                },
                {
                    "BoundingBox": {
                        "Width": 0.031183116137981415,
                        "Top": 0.5579366683959961,
                        "Left": 0.2853088080883026,
                        "Height": 0.03989990055561066
                    },
                    "Confidence": 81.0157470703125
                },
                {
                    "BoundingBox": {
                        "Width": 0.031113790348172188,
                        "Top": 0.5504819750785828,
                        "Left": 0.2580395042896271,
                        "Height": 0.056484755128622055
                    },
                    "Confidence": 56.13441467285156
                },
                {
                    "BoundingBox": {
                        "Width": 0.08586374670267105,
                        "Top": 0.5438792705535889,
                        "Left": 0.5128012895584106,
                        "Height": 0.08550430089235306
                    },
                    "Confidence": 52.37760925292969
```

```
                }
            ],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Car"
        },
        {
            "Instances": [],
            "Confidence": 98.9914321899414,
            "Parents": [],
            "Name": "Human"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.19360728561878204,
                        "Top": 0.35072067379951477,
                        "Left": 0.43734854459762573,
                        "Height": 0.2742200493812561
                    },
                    "Confidence": 98.9914321899414
                },
                {
                    "BoundingBox": {
                        "Width": 0.03801717236638069,
                        "Top": 0.5010883808135986,
                        "Left": 0.9155802130699158,
                        "Height": 0.06597328186035156
                    },
                    "Confidence": 85.02790832519531
                }
            ],
            "Confidence": 98.9914321899414,
            "Parents": [],
            "Name": "Person"
        },
```

```
        {
            "Instances": [],
            "Confidence": 93.24951934814453,
            "Parents": [],
            "Name": "Machine"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.03561960905790329,
                        "Top": 0.6468243598937988,
                        "Left": 0.7850857377052307,
                        "Height": 0.08878646790981293
                    },
                    "Confidence": 93.24951934814453
                },
                {
                    "BoundingBox": {
                        "Width": 0.02217046171426773,
                        "Top": 0.6149078607559204,
                        "Left": 0.04757237061858177,
                        "Height": 0.07136218994855881
                    },
                    "Confidence": 91.5025863647461
                },
                {
                    "BoundingBox": {
                        "Width": 0.016197510063648224,
                        "Top": 0.6274210214614868,
                        "Left": 0.6472989320755005,
                        "Height": 0.04955997318029404
                    },
                    "Confidence": 85.14686584472656
                },
                {
                    "BoundingBox": {
                        "Width": 0.020207518711686134,
                        "Top": 0.6348286867141724,
                        "Left": 0.7295016646385193,
                        "Height": 0.07059963047504425
                    },
                    "Confidence": 83.34547424316406
                },
```

```
                {
                    "BoundingBox": {
                        "Width": 0.020280985161662102,
                        "Top": 0.6171894669532776,
                        "Left": 0.08744934946298599,
                        "Height": 0.05297485366463661
                    },
                    "Confidence": 79.9981460571289
                },
                {
                    "BoundingBox": {
                        "Width": 0.018318990245461464,
                        "Top": 0.623889148235321,
                        "Left": 0.6836880445480347,
                        "Height": 0.06730121374130249
                    },
                    "Confidence": 78.87144470214844
                },
                {
                    "BoundingBox": {
                        "Width": 0.021310249343514442,
                        "Top": 0.6167286038398743,
                        "Left": 0.004064912907779217,
                        "Height": 0.08317798376083374
                    },
                    "Confidence": 75.89361572265625
                },
                {
                    "BoundingBox": {
                        "Width": 0.03604431077837944,
                        "Top": 0.7030032277107239,
                        "Left": 0.9254803657531738,
                        "Height": 0.04569442570209503
                    },
                    "Confidence": 64.402587890625
                },
                {
                    "BoundingBox": {
                        "Width": 0.009834849275648594,
                        "Top": 0.5821820497512817,
                        "Left": 0.28094568848609924,
                        "Height": 0.01964157074689865
                    },
                    "Confidence": 62.79907989501953
```

```
                },
                {
                    "BoundingBox": {
                        "Width": 0.01475677452981472,
                        "Top": 0.6137543320655823,
                        "Left": 0.5950819253921509,
                        "Height": 0.039063986390829086
                    },
                    "Confidence": 59.40483474731445
                }
            ],
            "Confidence": 93.24951934814453,
            "Parents": [
                {
                    "Name": "Machine"
                }
            ],
            "Name": "Wheel"
        },
        {
            "Instances": [],
            "Confidence": 92.61514282226562,
            "Parents": [],
            "Name": "Road"
        },
        {
            "Instances": [],
            "Confidence": 92.37877655029297,
            "Parents": [
                {
                    "Name": "Person"
                }
            ],
            "Name": "Sport"
        },
        {
            "Instances": [],
            "Confidence": 92.37877655029297,
            "Parents": [
                {
                    "Name": "Person"
                }
            ],
            "Name": "Sports"
```

```
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.12326609343290329,
                        "Top": 0.6332163214683533,
                        "Left": 0.44815489649772644,
                        "Height": 0.058117982000112534
                    },
                    "Confidence": 92.37877655029297
                }
            ],
            "Confidence": 92.37877655029297,
            "Parents": [
                {
                    "Name": "Person"
                },
                {
                    "Name": "Sport"
                }
            ],
            "Name": "Skateboard"
        },
        {
            "Instances": [],
            "Confidence": 90.62931060791016,
            "Parents": [
                {
                    "Name": "Person"
                }
            ],
            "Name": "Pedestrian"
        },
        {
            "Instances": [],
            "Confidence": 88.81334686279297,
            "Parents": [],
            "Name": "Asphalt"
        },
        {
            "Instances": [],
            "Confidence": 88.81334686279297,
            "Parents": [],
```

```
            "Name": "Tarmac"
        },
        {
            "Instances": [],
            "Confidence": 88.23201751708984,
            "Parents": [],
            "Name": "Path"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [],
            "Name": "Urban"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "Town"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [],
            "Name": "Building"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
```

```
            "Name": "City"
        },
        {
            "Instances": [],
            "Confidence": 78.37934875488281,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Parking Lot"
        },
        {
            "Instances": [],
            "Confidence": 78.37934875488281,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Parking"
        },
        {
            "Instances": [],
            "Confidence": 74.37590026855469,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                },
```

```
                {
                    "Name": "City"
                }
            ],
            "Name": "Downtown"
        },
        {
            "Instances": [],
            "Confidence": 69.84622955322266,
            "Parents": [
                {
                    "Name": "Road"
                }
            ],
            "Name": "Intersection"
        },
        {
            "Instances": [],
            "Confidence": 57.68518829345703,
            "Parents": [
                {
                    "Name": "Sports Car"
                },
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Coupe"
        },
        {
            "Instances": [],
            "Confidence": 57.68518829345703,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
```

```
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Sports Car"
        },
        {
            "Instances": [],
            "Confidence": 56.59492111206055,
            "Parents": [
                {
                    "Name": "Path"
                }
            ],
            "Name": "Sidewalk"
        },
        {
            "Instances": [],
            "Confidence": 56.59492111206055,
            "Parents": [
                {
                    "Name": "Path"
                }
            ],
            "Name": "Pavement"
        },
        {
            "Instances": [],
            "Confidence": 55.58770751953125,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "Neighborhood"
        }
    ],
    "LabelModelVersion": "2.0"
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的偵測影像中的標籤。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DetectLabels中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectLabels {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <sourceImage>
```

```
                Where:
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectImageLabels(rekClient, sourceImage);
        rekClient.close();
    }

    public static void detectImageLabels(RekognitionClient rekClient, String
 sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            DetectLabelsRequest detectLabelsRequest =
 DetectLabelsRequest.builder()
                    .image(souImage)
                    .maxLabels(10)
                    .build();

            DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label : labels) {
                System.out.println(label.name() + ": " +
 label.confidence().toString());
```

```
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DetectLabels中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun detectImageLabels(sourceImage: String) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }
    val request =
        DetectLabelsRequest {
            image = souImage
            maxLabels = 10
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectLabels(request)
        response.labels?.forEach { label ->
            println("${label.name} : ${label.confidence}")
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK DetectLabels中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.

        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels
```

```
            )
            labels = [RekognitionLabel(label) for label in response["Labels"]]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DetectLabels中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DetectModerationLabels配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DetectModerationLabels。

如需詳細資訊，請參閱偵測不適合的映像。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect unsafe content in a
/// JPEG or PNG format image.
/// </summary>
```

```
   public class DetectModerationLabels
   {
       public static async Task Main(string[] args)
       {
           string photo = "input.jpg";
           string bucket = "bucket";

           var rekognitionClient = new AmazonRekognitionClient();

           var detectModerationLabelsRequest = new
DetectModerationLabelsRequest()
           {
               Image = new Image()
               {
                   S3Object = new S3Object()
                   {
                       Name = photo,
                       Bucket = bucket,
                   },
               },
               MinConfidence = 60F,
           };

           try
           {
               var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
               Console.WriteLine("Detected labels for " + photo);
               foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
               {
                   Console.WriteLine($"Label: {label.Name}");
                   Console.WriteLine($"Confidence: {label.Confidence}");
                   Console.WriteLine($"Parent: {label.ParentName}");
               }
           }
           catch (Exception ex)
           {
               Console.WriteLine(ex.Message);
           }
       }
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考中的DetectModeration標籤。

CLI

AWS CLI

偵測影像中不安全的內容

下列detect-moderation-labels命令會偵測存放在 Amazon S3 儲存貯體中的指定映像中的不安全內容。

```
aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=gun.jpg}"
```

輸出：

```
{
    "ModerationModelVersion": "3.0",
    "ModerationLabels": [
        {
            "Confidence": 97.29618072509766,
            "ParentName": "Violence",
            "Name": "Weapon Violence"
        },
        {
            "Confidence": 97.29618072509766,
            "ParentName": "",
            "Name": "Violence"
        }
    ]
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的偵測不安全的映像。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考中的DetectModeration標籤。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsRequest;
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.ModerationLabel;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectModerationLabels {

    public static void main(String[] args) {
        final String usage = """

                Usage:    <sourceImage>

                Where:
```

```
                    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """";

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectModLabels(rekClient, sourceImage);
        rekClient.close();
    }

    public static void detectModLabels(RekognitionClient rekClient, String
 sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            DetectModerationLabelsRequest moderationLabelsRequest =
 DetectModerationLabelsRequest.builder()
                    .image(souImage)
                    .minConfidence(60F)
                    .build();

            DetectModerationLabelsResponse moderationLabelsResponse = rekClient
                    .detectModerationLabels(moderationLabelsRequest);
            List<ModerationLabel> labels =
 moderationLabelsResponse.moderationLabels();
            System.out.println("Detected labels for image");
            for (ModerationLabel label : labels) {
                System.out.println("Label: " + label.name()
                        + "\n Confidence: " + label.confidence().toString() + "%"
                        + "\n Parent:" + label.parentName());
            }
```

```
        } catch (RekognitionException | FileNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的DetectModeration標籤。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun detectModLabels(sourceImage: String) {
    val myImage =
        Image {
            this.bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectModerationLabelsRequest {
            image = myImage
            minConfidence = 60f
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectModerationLabels(request)
        response.moderationLabels?.forEach { label ->
            println("Label: ${label.name} - Confidence: ${label.confidence} %
 Parent: ${label.parentName}")
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK 中的DetectModeration標籤以獲取 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_moderation_labels(self):
        """
        Detects moderation labels in the image. Moderation labels identify
 content
        that may be inappropriate for some audiences.
```

```
        :return: The list of moderation labels found in the image.
        """
        try:
            response = self.rekognition_client.detect_moderation_labels(
                Image=self.image
            )
            labels = [
                RekognitionModerationLabel(label)
                for label in response["ModerationLabels"]
            ]
            logger.info(
                "Found %s moderation labels in %s.", len(labels), self.image_name
            )
        except ClientError:
            logger.exception(
                "Couldn't detect moderation labels in %s.", self.image_name
            )
            raise
        else:
            return labels
```

- 如需 API 詳細資訊，請參閱AWS 開發套件中的DetectModeration標籤 (Boto3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭**DetectText**配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DetectText。

如需更多資訊，請參閱偵測映像中的文字。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect text in an image. The
/// example was created using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class DetectText
{
    public static async Task Main()
    {
        string photo = "Dad_photographer.jpg"; // "input.jpg";
        string bucket = "igsmiths3photos"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
        };

        try
```

```
            {
                DetectTextResponse detectTextResponse = await
 rekognitionClient.DetectTextAsync(detectTextRequest);
                Console.WriteLine($"Detected lines and words for {photo}");
                detectTextResponse.TextDetections.ForEach(text =>
                {
                    Console.WriteLine($"Detected: {text.DetectedText}");
                    Console.WriteLine($"Confidence: {text.Confidence}");
                    Console.WriteLine($"Id : {text.Id}");
                    Console.WriteLine($"Parent Id: {text.ParentId}");
                    Console.WriteLine($"Type: {text.Type}");
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DetectText中的。

CLI

AWS CLI

偵測影像中的文字

以下`detect-text`命令檢測指定圖像中的文本。

```
aws rekognition detect-text \
    --image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"ExamplePicture.jpg"}}'
```

輸出：

```
{
    "TextDetections": [
        {
            "Geometry": {
                "BoundingBox": {
```

```
                    "Width": 0.24624845385551453,
                    "Top": 0.28288066387176514,
                    "Left": 0.391388863325119,
                    "Height": 0.022687450051307678
                },
                "Polygon": [
                    {
                        "Y": 0.28288066387176514,
                        "X": 0.391388863325119
                    },
                    {
                        "Y": 0.2826388478279114,
                        "X": 0.6376373171806335
                    },
                    {
                        "Y": 0.30532628297805786,
                        "X": 0.637677013874054
                    },
                    {
                        "Y": 0.305568128824234,
                        "X": 0.39142853021621704
                    }
                ]
            },
            "Confidence": 94.35709381103516,
            "DetectedText": "ESTD 1882",
            "Type": "LINE",
            "Id": 0
        },
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933889865875244,
                    "Top": 0.32603850960731506,
                    "Left": 0.34534579515457153,
                    "Height": 0.07126858830451965
                },
                "Polygon": [
                    {
                        "Y": 0.32603850960731506,
                        "X": 0.34534579515457153
                    },
                    {
                        "Y": 0.32633158564567566,
```

```
                        "X": 0.684684693813324
                    },
                    {
                        "Y": 0.3976001739501953,
                        "X": 0.684575080871582
                    },
                    {
                        "Y": 0.3973070979118347,
                        "X": 0.345236212015152
                    }
                ]
            },
            "Confidence": 99.95779418945312,
            "DetectedText": "BRAINS",
            "Type": "LINE",
            "Id": 1
        },
        {
            "Confidence": 97.22098541259766,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.061079490929841995,
                    "Top": 0.2843210697174072,
                    "Left": 0.391391396522522,
                    "Height": 0.021029088646173477
                },
                "Polygon": [
                    {
                        "Y": 0.2843210697174072,
                        "X": 0.391391396522522
                    },
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.4524524509906769
                    },
                    {
                        "Y": 0.3038259446620941,
                        "X": 0.4534534513950348
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.3923923969268799
                    }
                ]
```

```
            },
            "DetectedText": "ESTD",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 2
        },
        {
            "Confidence": 91.49320983886719,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.07007007300853729,
                    "Top": 0.2828207015991211,
                    "Left": 0.5675675868988037,
                    "Height": 0.02250562608242035
                },
                "Polygon": [
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.5675675868988037
                    },
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.5675675868988037
                    }
                ]
            },
            "DetectedText": "1882",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 3
        },
        {
            "Confidence": 99.95779418945312,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933934569358826,
```

```
                    "Top": 0.32633158564567566,
                    "Left": 0.3453453481197357,
                    "Height": 0.07127484679222107
                },
                "Polygon": [
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.3453453481197357
                    },
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.684684693813324
                    },
                    {
                        "Y": 0.39759939908981323,
                        "X": 0.6836836934089661
                    },
                    {
                        "Y": 0.39684921503067017,
                        "X": 0.3453453481197357
                    }
                ]
            },
            "DetectedText": "BRAINS",
            "ParentId": 1,
            "Type": "WORD",
            "Id": 4
        }
    ]
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[DetectText](#)中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DetectTextRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectTextResponse;
import software.amazon.awssdk.services.rekognition.model.TextDetection;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectText {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <sourceImage>

                Where:
                   sourceImage - The path to the image that contains text (for
 example, C:\\AWS\\pic1.png).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();
```

```
        detectTextLabels(rekClient, sourceImage);
        rekClient.close();
    }

    public static void detectTextLabels(RekognitionClient rekClient, String
 sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            DetectTextRequest textRequest = DetectTextRequest.builder()
                    .image(souImage)
                    .build();

            DetectTextResponse textResponse = rekClient.detectText(textRequest);
            List<TextDetection> textCollection = textResponse.textDetections();
            System.out.println("Detected lines and words");
            for (TextDetection text : textCollection) {
                System.out.println("Detected: " + text.detectedText());
                System.out.println("Confidence: " +
 text.confidence().toString());
                System.out.println("Id : " + text.id());
                System.out.println("Parent Id: " + text.parentId());
                System.out.println("Type: " + text.type());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DetectText中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun detectTextLabels(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectTextRequest {
            image = souImage
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectText(request)
        response.textDetections?.forEach { text ->
            println("Detected: ${text.detectedText}")
            println("Confidence: ${text.confidence}")
            println("Id: ${text.id}")
            println("Parent Id:  ${text.parentId}")
            println("Type: ${text.type}")
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK DetectText中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [RekognitionText(text) for text in
 response["TextDetections"]]
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
```

```
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DetectText中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DisassociateFaces配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DisassociateFaces。

CLI

AWS CLI

```
aws rekognition disassociate-faces --face-ids list-of-face-ids
  --user-id user-id --collection-id collection-name --region region-name
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DisassociateFaces中的。

Python

適用於 Python (Boto3) 的 SDK

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def disassociate_faces(collection_id, user_id, face_ids):
    """
    Disassociate stored faces within collection to the given user
```

```
    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to disassociate faces from
    :param face_ids: The list of face IDs to be disassociated from the given user

    :return: response of AssociateFaces API
    """
    logger.info(f'Disssociating faces from user: {user_id}, {face_ids}')
    try:
        response = client.disassociate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- disassociated {len(response["DisassociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to disassociate faces from the given user")
        raise
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    disassociate_faces(collection_id, user_id, face_ids)

if __name__ == "__main__":
    main()
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DisassociateFaces中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetCelebrityInfo配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetCelebrityInfo。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to retrieve information about the
    /// celebrity identified by the supplied celebrity Id.
    /// </summary>
    public class CelebrityInfo
    {
        public static async Task Main()
        {
            string celebId = "nnnnnnnn";

            var rekognitionClient = new AmazonRekognitionClient();

            var celebrityInfoRequest = new GetCelebrityInfoRequest
            {
                Id = celebId,
            };

            Console.WriteLine($"Getting information for celebrity: {celebId}");

            var celebrityInfoResponse = await
rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

            // Display celebrity information.
            Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
            Console.WriteLine("Further information (if available):");
            celebrityInfoResponse.Urls.ForEach(url =>
            {
```

```
                Console.WriteLine(url);
            });
        }
    }
```

- 有關 API 詳細GetCelebrity信息，請參閱 AWS SDK for .NET API 參考中的信息。

CLI

AWS CLI

獲取有關名人的信息

以下get-celebrity-info命令顯示有關指定名人的信息。id參數來自先前的呼叫recognize-celebrities。

```
aws rekognition get-celebrity-info --id nnnnnnn
```

輸出：

```
{
    "Name": "Celeb A",
    "Urls": [
        "www.imdb.com/name/aaaaaaaaa"
    ]
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的取得名人相關資訊。

- 如需 API 詳細GetCelebrity資訊，請參閱AWS CLI 命令參考中的資訊。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭IndexFaces配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用IndexFaces。

如需詳細資訊，請參閱將人臉新增至集合。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces in an image
/// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
/// bucket and then adds the information to a collection.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "doc-example-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var image = new Image
        {
            S3Object = new S3Object
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var indexFacesRequest = new IndexFacesRequest
        {
```

```
                Image = image,
                CollectionId = collectionId,
                ExternalImageId = photo,
                DetectionAttributes = new List<string>() { "ALL" },
            };

            IndexFacesResponse indexFacesResponse = await
 rekognitionClient.IndexFacesAsync(indexFacesRequest);

            Console.WriteLine($"{photo} added");
            foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
            {
                Console.WriteLine($"Face detected: Faceid is
 {faceRecord.Face.FaceId}");
            }
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考IndexFaces中的。

CLI

AWS CLI

將面加入至集合的步驟

以下指index-faces令會將在影像中找到的面加入至指定的集合。

```
aws rekognition index-faces \
    --image '{"S3Object":{"Bucket":"MyVideoS3Bucket","Name":"MyPicture.jpg"}}' \
    --collection-id MyCollection \
    --max-faces 1 \
    --quality-filter "AUTO" \
    --detection-attributes "ALL" \
    --external-image-id "MyPicture.jpg"
```

輸出：

```
{
    "FaceRecords": [
        {
```

```
            "FaceDetail": {
                "Confidence": 99.993408203125,
                "Eyeglasses": {
                    "Confidence": 99.11750030517578,
                    "Value": false
                },
                "Sunglasses": {
                    "Confidence": 99.98249053955078,
                    "Value": false
                },
                "Gender": {
                    "Confidence": 99.92769622802734,
                    "Value": "Male"
                },
                "Landmarks": [
                    {
                        "Y": 0.26750367879867554,
                        "X": 0.6202793717384338,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.26642778515815735,
                        "X": 0.6787431836128235,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.31361380219459534,
                        "X": 0.6421601176261902,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.3495299220085144,
                        "X": 0.6216195225715637,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.35194727778434753,
                        "X": 0.669899046421051,
                        "Type": "mouthRight"
                    },
                    {
                        "Y": 0.26844894886016846,
                        "X": 0.6210268139839172,
                        "Type": "leftPupil"
```

```
                    },
                    {
                        "Y": 0.26707562804222107,
                        "X": 0.6817160844802856,
                        "Type": "rightPupil"
                    },
                    {
                        "Y": 0.24834522604942322,
                        "X": 0.6018546223640442,
                        "Type": "leftEyeBrowLeft"
                    },
                    {
                        "Y": 0.24397172033786774,
                        "X": 0.6172008514404297,
                        "Type": "leftEyeBrowUp"
                    },
                    {
                        "Y": 0.24677404761314392,
                        "X": 0.6339119076728821,
                        "Type": "leftEyeBrowRight"
                    },
                    {
                        "Y": 0.24582654237747192,
                        "X": 0.6619398593902588,
                        "Type": "rightEyeBrowLeft"
                    },
                    {
                        "Y": 0.23973053693771362,
                        "X": 0.6804757118225098,
                        "Type": "rightEyeBrowUp"
                    },
                    {
                        "Y": 0.24441994726657867,
                        "X": 0.6978968977928162,
                        "Type": "rightEyeBrowRight"
                    },
                    {
                        "Y": 0.2695908546447754,
                        "X": 0.6085202693939209,
                        "Type": "leftEyeLeft"
                    },
                    {
                        "Y": 0.26716896891593933,
                        "X": 0.6315826177597046,
```

```
                        "Type": "leftEyeRight"
                    },
                    {
                        "Y": 0.26289820671081543,
                        "X": 0.6202316880226135,
                        "Type": "leftEyeUp"
                    },
                    {
                        "Y": 0.27123287320137024,
                        "X": 0.6205548048019409,
                        "Type": "leftEyeDown"
                    },
                    {
                        "Y": 0.2668408751487732,
                        "X": 0.6663622260093689,
                        "Type": "rightEyeLeft"
                    },
                    {
                        "Y": 0.26741549372673035,
                        "X": 0.6910083889961243,
                        "Type": "rightEyeRight"
                    },
                    {
                        "Y": 0.2614026665687561,
                        "X": 0.6785826086997986,
                        "Type": "rightEyeUp"
                    },
                    {
                        "Y": 0.27075251936912537,
                        "X": 0.6789616942405701,
                        "Type": "rightEyeDown"
                    },
                    {
                        "Y": 0.3211299479007721,
                        "X": 0.6324167847633362,
                        "Type": "noseLeft"
                    },
                    {
                        "Y": 0.32276326417922974,
                        "X": 0.6558475494384766,
                        "Type": "noseRight"
                    },
                    {
                        "Y": 0.34385165572166443,
```

```
                        "X": 0.6444970965385437,
                        "Type": "mouthUp"
                    },
                    {
                        "Y": 0.3671635091304779,
                        "X": 0.6459195017814636,
                        "Type": "mouthDown"
                    }
                ],
                "Pose": {
                    "Yaw": -9.54541015625,
                    "Roll": -0.5709401965141296,
                    "Pitch": 0.6045494675636292
                },
                "Emotions": [
                    {
                        "Confidence": 39.90074157714844,
                        "Type": "HAPPY"
                    },
                    {
                        "Confidence": 23.38753890991211,
                        "Type": "CALM"
                    },
                    {
                        "Confidence": 5.840933322906494,
                        "Type": "CONFUSED"
                    }
                ],
                "AgeRange": {
                    "High": 63,
                    "Low": 45
                },
                "EyesOpen": {
                    "Confidence": 99.80887603759766,
                    "Value": true
                },
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618015021085739,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770642817020416
                },
                "Smile": {
                    "Confidence": 99.69740295410156,
```

```
                    "Value": false
                },
                "MouthOpen": {
                    "Confidence": 99.97393798828125,
                    "Value": false
                },
                "Quality": {
                    "Sharpness": 95.54405975341797,
                    "Brightness": 63.867706298828125
                },
                "Mustache": {
                    "Confidence": 97.05007934570312,
                    "Value": false
                },
                "Beard": {
                    "Confidence": 87.34505462646484,
                    "Value": false
                }
            },
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618015021085739,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770642817020416
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.993408203125,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            }
        }
    ],
    "UnindexedFaces": [],
    "FaceModelVersion": "3.0",
    "OrientationCorrection": "ROTATE_0"
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的將人臉新增至集合。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考IndexFaces中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.IndexFacesResponse;
import software.amazon.awssdk.services.rekognition.model.IndexFacesRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.QualityFilter;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceRecord;
import software.amazon.awssdk.services.rekognition.model.UnindexedFace;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Reason;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AddFacesToCollection {
    public static void main(String[] args) {

        final String usage = """

                Usage:     <collectionId> <sourceImage>
```

```
                Where:
                    collectionName - The name of the collection.
                    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        addToCollection(rekClient, collectionId, sourceImage);
        rekClient.close();
    }

    public static void addToCollection(RekognitionClient rekClient, String
 collectionId, String sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            IndexFacesRequest facesRequest = IndexFacesRequest.builder()
                    .collectionId(collectionId)
                    .image(souImage)
                    .maxFaces(1)
                    .qualityFilter(QualityFilter.AUTO)
                    .detectionAttributes(Attribute.DEFAULT)
                    .build();

            IndexFacesResponse facesResponse =
 rekClient.indexFaces(facesRequest);
            System.out.println("Results for the image");
            System.out.println("\n Faces indexed:");
```

```
            List<FaceRecord> faceRecords = facesResponse.faceRecords();
            for (FaceRecord faceRecord : faceRecords) {
                System.out.println("  Face ID: " + faceRecord.face().faceId());
                System.out.println("  Location:" +
 faceRecord.faceDetail().boundingBox().toString());
            }

            List<UnindexedFace> unindexedFaces = facesResponse.unindexedFaces();
            System.out.println("Faces not indexed:");
            for (UnindexedFace unindexedFace : unindexedFaces) {
                System.out.println("  Location:" +
 unindexedFace.faceDetail().boundingBox().toString());
                System.out.println("  Reasons:");
                for (Reason reason : unindexedFace.reasons()) {
                    System.out.println("Reason:  " + reason);
                }
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考IndexFaces中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun addToCollection(
    collectionIdVal: String?,
    sourceImage: String,
) {
```

```
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        IndexFacesRequest {
            collectionId = collectionIdVal
            image = souImage
            maxFaces = 1
            qualityFilter = QualityFilter.Auto
            detectionAttributes = listOf(Attribute.Default)
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val facesResponse = rekClient.indexFaces(request)

        // Display the results.
        println("Results for the image")
        println("\n Faces indexed:")
        facesResponse.faceRecords?.forEach { faceRecord ->
            println("Face ID: ${faceRecord.face?.faceId}")
            println("Location: ${faceRecord.faceDetail?.boundingBox}")
        }

        println("Faces not indexed:")
        facesResponse.unindexedFaces?.forEach { unindexedFace ->
            println("Location: ${unindexedFace.faceDetail?.boundingBox}")
            println("Reasons:")

            unindexedFace.reasons?.forEach { reason ->
                println("Reason:  $reason")
            }
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK IndexFaces中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
```

```
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def index_faces(self, image, max_faces):
        """
        Finds faces in the specified image, indexes them, and stores them in the
        collection.

        :param image: The image to index.
        :param max_faces: The maximum number of faces to index.
        :return: A tuple. The first element is a list of indexed faces.
                 The second element is a list of faces that couldn't be indexed.
        """
        try:
            response = self.rekognition_client.index_faces(
                CollectionId=self.collection_id,
                Image=image.image,
                ExternalImageId=image.image_name,
                MaxFaces=max_faces,
                DetectionAttributes=["ALL"],
            )
            indexed_faces = [
                RekognitionFace({**face["Face"], **face["FaceDetail"]})
                for face in response["FaceRecords"]
            ]
            unindexed_faces = [
                RekognitionFace(face["FaceDetail"])
                for face in response["UnindexedFaces"]
            ]
            logger.info(
                "Indexed %s faces in %s. Could not index %s faces.",
                len(indexed_faces),
                image.image_name,
                len(unindexed_faces),
            )
        except ClientError:
            logger.exception("Couldn't index faces in image %s.",
image.image_name)
            raise
        else:
            return indexed_faces, unindexed_faces
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件IndexFaces中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭ListCollections配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用ListCollections。

如需詳細資訊，請參閱列出的集合。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses Amazon Rekognition to list the collection IDs in the
/// current account.
/// </summary>
public class ListCollections
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        Console.WriteLine("Listing collections");
        int limit = 10;
```

```
            var listCollectionsRequest = new ListCollectionsRequest
            {
                MaxResults = limit,
            };

            var listCollectionsResponse = new ListCollectionsResponse();

            do
            {
                if (listCollectionsResponse is not null)
                {
                    listCollectionsRequest.NextToken =
listCollectionsResponse.NextToken;
                }

                listCollectionsResponse = await
rekognitionClient.ListCollectionsAsync(listCollectionsRequest);

                listCollectionsResponse.CollectionIds.ForEach(id =>
                {
                    Console.WriteLine(id);
                });
            }
            while (listCollectionsResponse.NextToken is not null);
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[ListCollections](#)中的。

CLI

AWS CLI

列出可用集合的步驟

以下`list-collections`命令列出了 AWS 帳戶中可用的集合。

```
aws rekognition list-collections
```

輸出：

```
{
    "FaceModelVersions": [
        "2.0",
        "3.0",
        "3.0",
        "3.0",
        "4.0",
        "1.0",
        "3.0",
        "4.0",
        "4.0",
        "4.0"
    ],
    "CollectionIds": [
        "MyCollection1",
        "MyCollection2",
        "MyCollection3",
        "MyCollection4",
        "MyCollection5",
        "MyCollection6",
        "MyCollection7",
        "MyCollection8",
        "MyCollection9",
        "MyCollection10"
    ]
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的列出系列。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ListCollections中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsRequest;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListCollections {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Listing collections");
        listAllCollections(rekClient);
        rekClient.close();
    }

    public static void listAllCollections(RekognitionClient rekClient) {
        try {
            ListCollectionsRequest listCollectionsRequest =
 ListCollectionsRequest.builder()
                    .maxResults(10)
                    .build();

            ListCollectionsResponse response =
 rekClient.listCollections(listCollectionsRequest);
            List<String> collectionIds = response.collectionIds();
            for (String resultId : collectionIds) {
                System.out.println(resultId);
            }

        } catch (RekognitionException e) {
```

```
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListCollections中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun listAllCollections() {
    val request =
        ListCollectionsRequest {
            maxResults = 10
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listCollections(request)
        response.collectionIds?.forEach { resultId ->
            println(resultId)
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK ListCollections中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
 API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client


    def list_collections(self, max_results):
        """
        Lists collections for the current account.

        :param max_results: The maximum number of collections to return.
        :return: The list of collections for the current account.
        """
        try:
            response =
 self.rekognition_client.list_collections(MaxResults=max_results)
            collections = [
                RekognitionCollection({"CollectionId": col_id},
 self.rekognition_client)
                for col_id in response["CollectionIds"]
            ]
        except ClientError:
```

```
            logger.exception("Couldn't list collections.")
            raise
        else:
            return collections
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListCollections中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭ListFaces配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用ListFaces。

如需更多資訊，請參閱集合中列出的人臉。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to retrieve the list of faces
/// stored in a collection.
/// </summary>
public class ListFaces
{
    public static async Task Main()
```

```
        {
            string collectionId = "MyCollection2";

            var rekognitionClient = new AmazonRekognitionClient();

            var listFacesResponse = new ListFacesResponse();
            Console.WriteLine($"Faces in collection {collectionId}");

            var listFacesRequest = new ListFacesRequest
            {
                CollectionId = collectionId,
                MaxResults = 1,
            };

            do
            {
                listFacesResponse = await
 rekognitionClient.ListFacesAsync(listFacesRequest);
                listFacesResponse.Faces.ForEach(face =>
                {
                    Console.WriteLine(face.FaceId);
                });

                listFacesRequest.NextToken = listFacesResponse.NextToken;
            }
            while (!string.IsNullOrEmpty(listFacesResponse.NextToken));
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考ListFaces中的。

CLI

AWS CLI

列示集合中面孔的步驟

以下`list-faces`命令列出了指定集合中的面。

```
aws rekognition list-faces \
```

```
    --collection-id MyCollection
```

輸出：

```
{
    "FaceModelVersion": "3.0",
    "Faces": [
        {
            "BoundingBox": {
                "Width": 0.5216310024261475,
                "Top": 0.3256250023841858,
                "Left": 0.13394300639629364,
                "Height": 0.3918749988079071
            },
            "FaceId": "0040279c-0178-436e-b70a-e61b074e96b0",
            "ExternalImageId": "image1.jpg",
            "Confidence": 100.0,
            "ImageId": "f976e487-3719-5e2d-be8b-ea2724c26991"
        },
        {
            "BoundingBox": {
                "Width": 0.5074880123138428,
                "Top": 0.3774999976158142,
                "Left": 0.18302799761295319,
                "Height": 0.3812499940395355
            },
            "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
            "ExternalImageId": "image2.jpg",
            "Confidence": 99.99930572509766,
            "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
        },
        {
            "BoundingBox": {
                "Width": 0.5574039816856384,
                "Top": 0.37187498807907104,
                "Left": 0.14559100568294525,
                "Height": 0.4181250035762787
            },
            "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
            "ExternalImageId": "image3.jpg",
            "Confidence": 99.99960327148438,
            "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
        },
```

```
        {
            "BoundingBox": {
                "Width": 0.18562500178813934,
                "Top": 0.1618019938468933,
                "Left": 0.5575000047683716,
                "Height": 0.24770599603652954
            },
            "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
            "ExternalImageId": "image4.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
        },
        {
            "BoundingBox": {
                "Width": 0.5307819843292236,
                "Top": 0.2862499952316284,
                "Left": 0.1564060002565384,
                "Height": 0.3987500071525574
            },
            "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
            "ExternalImageId": "image5.jpg",
            "Confidence": 99.99970245361328,
            "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
        },
        {
            "BoundingBox": {
                "Width": 0.5773710012435913,
                "Top": 0.34437501430511475,
                "Left": 0.12396000325679779,
                "Height": 0.4337500035762787
            },
            "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
            "ExternalImageId": "image6.jpg",
            "Confidence": 100.0,
            "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
        },
        {
            "BoundingBox": {
                "Width": 0.5349419713020325,
                "Top": 0.29124999046325684,
                "Left": 0.16389399766921997,
                "Height": 0.40187498927116394
            },
            "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
```

```
            "ExternalImageId": "image7.jpg",
            "Confidence": 99.99979400634766,
            "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
        },
        {
            "BoundingBox": {
                "Width": 0.41499999165534973,
                "Top": 0.09187500178813934,
                "Left": 0.28083300590515137,
                "Height": 0.3112500011920929
            },
            "FaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
            "ExternalImageId": "image8.jpg",
            "Confidence": 99.99769592285156,
            "ImageId": "a294da46-2cb1-5cc4-9045-61d7ca567662"
        },
        {
            "BoundingBox": {
                "Width": 0.48166701197624207,
                "Top": 0.20999999344348907,
                "Left": 0.21250000596046448,
                "Height": 0.36125001311302185
            },
            "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
            "ExternalImageId": "image9.jpg",
            "Confidence": 99.99949645996094,
            "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
        },
        {
            "BoundingBox": {
                "Width": 0.18562500178813934,
                "Top": 0.1618019938468933,
                "Left": 0.5575000047683716,
                "Height": 0.24770599603652954
            },
            "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
            "ExternalImageId": "image10.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
        }
    ]
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的列出集合中的臉孔。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ListFaces中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Face;
import software.amazon.awssdk.services.rekognition.model.ListFacesRequest;
import software.amazon.awssdk.services.rekognition.model.ListFacesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListFacesInCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId>

                Where:
                   collectionId - The name of the collection.\s
                """;

        if (args.length < 1) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Faces in collection " + collectionId);
        listFacesCollection(rekClient, collectionId);
        rekClient.close();
    }

    public static void listFacesCollection(RekognitionClient rekClient, String
 collectionId) {
        try {
            ListFacesRequest facesRequest = ListFacesRequest.builder()
                    .collectionId(collectionId)
                    .maxResults(10)
                    .build();

            ListFacesResponse facesResponse = rekClient.listFaces(facesRequest);
            List<Face> faces = facesResponse.faces();
            for (Face face : faces) {
                System.out.println("Confidence level there is a face: " +
 face.confidence());
                System.out.println("The face Id value is " + face.faceId());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListFaces中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun listFacesCollection(collectionIdVal: String?) {
    val request =
        ListFacesRequest {
            collectionId = collectionIdVal
            maxResults = 10
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listFaces(request)
        response.faces?.forEach { face ->
            println("Confidence level there is a face: ${face.confidence}")
            println("The face Id value is ${face.faceId}")
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK ListFaces中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollection:
```

```
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def list_faces(self, max_results):
        """
        Lists the faces currently indexed in the collection.

        :param max_results: The maximum number of faces to return.
        :return: The list of faces in the collection.
        """
        try:
```

```
            response = self.rekognition_client.list_faces(
                CollectionId=self.collection_id, MaxResults=max_results
            )
            faces = [RekognitionFace(face) for face in response["Faces"]]
            logger.info(
                "Found %s faces in collection %s.", len(faces),
self.collection_id
            )
        except ClientError:
            logger.exception(
                "Couldn't list faces in collection %s.", self.collection_id
            )
            raise
        else:
            return faces
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListFaces中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭RecognizeCelebrities配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用RecognizeCelebrities。

如需詳細資訊，請參閱在映像中辨識名人。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.IO;
```

```
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to identify celebrities in a photo.
    /// </summary>
    public class CelebritiesInImage
    {
        public static async Task Main(string[] args)
        {
            string photo = "moviestars.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

            var img = new Amazon.Rekognition.Model.Image();
            byte[] data = null;
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
            }
            catch (Exception)
            {
                Console.WriteLine($"Failed to load file {photo}");
                return;
            }

            img.Bytes = new MemoryStream(data);
            recognizeCelebritiesRequest.Image = img;

            Console.WriteLine($"Looking for celebrities in image {photo}\n");

            var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);


Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
            recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
```

```
            {
                Console.WriteLine($"Celebrity recognized: {celeb.Name}");
                Console.WriteLine($"Celebrity ID: {celeb.Id}");
                BoundingBox boundingBox = celeb.Face.BoundingBox;
                Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
                Console.WriteLine("Further information (if available):");
                celeb.Urls.ForEach(url =>
                {
                    Console.WriteLine(url);
                });
            });


Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count}
face(s) were unrecognized.");
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考RecognizeCelebrities中的。

CLI

AWS CLI

辨識影像中的名人

以下recognize-celebrities命令可識別存放在 Amazon S3 儲存貯體中的指定映像中的名人。：

```
aws rekognition recognize-celebrities \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=moviestars.jpg}"
```

輸出：

```
{
    "UnrecognizedFaces": [
        {
            "BoundingBox": {
                "Width": 0.14416666328907013,
```

```
                "Top": 0.07777778059244156,
                "Left": 0.625,
                "Height": 0.2746031880378723
            },
            "Confidence": 99.9990234375,
            "Pose": {
                "Yaw": 10.80408763885498,
                "Roll": -12.761146545410156,
                "Pitch": 10.96889877319336
            },
            "Quality": {
                "Sharpness": 94.1185531616211,
                "Brightness": 79.18367004394531
            },
            "Landmarks": [
                {
                    "Y": 0.18220913410186768,
                    "X": 0.6702951788902283,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.16337193548679352,
                    "X": 0.7188183665275574,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.20739148557186127,
                    "X": 0.7055801749229431,
                    "Type": "nose"
                },
                {
                    "Y": 0.2889308035373688,
                    "X": 0.687512218952179,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.2706988751888275,
                    "X": 0.7250053286552429,
                    "Type": "mouthRight"
                }
            ]
        }
    ],
    "CelebrityFaces": [
```

```
        {
            "MatchConfidence": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.14000000059604645,
                    "Top": 0.1190476194024086,
                    "Left": 0.82833331823349,
                    "Height": 0.2666666805744171
                },
                "Confidence": 99.99359130859375,
                "Pose": {
                    "Yaw": -10.509642601013184,
                    "Roll": -14.51749324798584,
                    "Pitch": 13.799399375915527
                },
                "Quality": {
                    "Sharpness": 78.74752044677734,
                    "Brightness": 42.201324462890625
                },
                "Landmarks": [
                    {
                        "Y": 0.2290833294391632,
                        "X": 0.8709492087364197,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.20639978349208832,
                        "X": 0.9153988361358643,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.25417643785476685,
                        "X": 0.8907724022865295,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.32729196548461914,
                        "X": 0.8876466155052185,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.3115464746952057,
                        "X": 0.9238573312759399,
                        "Type": "mouthRight"
```

```
                    }
                ]
            },
            "Name": "Celeb A",
            "Urls": [
                "www.imdb.com/name/aaaaaaaaa"
            ],
            "Id": "1111111"
        },
        {
            "MatchConfidence": 97.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.13333334028720856,
                    "Top": 0.24920634925365448,
                    "Left": 0.4449999928474426,
                    "Height": 0.2539682686328888
                },
                "Confidence": 99.99979400634766,
                "Pose": {
                    "Yaw": 6.557040691375732,
                    "Roll": -7.316643714904785,
                    "Pitch": 9.272967338562012
                },
                "Quality": {
                    "Sharpness": 83.23492431640625,
                    "Brightness": 78.83267974853516
                },
                "Landmarks": [
                    {
                        "Y": 0.3625510632991791,
                        "X": 0.48898839950561523,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.35366007685661316,
                        "X": 0.5313721299171448,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.3894785940647125,
                        "X": 0.5173314809799194,
                        "Type": "nose"
                    },
```

```
                    {
                        "Y": 0.44889405369758606,
                        "X": 0.5020005702972412,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.4408611059188843,
                        "X": 0.5351271629333496,
                        "Type": "mouthRight"
                    }
                ]
            },
            "Name": "Celeb B",
            "Urls": [
                "www.imdb.com/name/bbbbbbbbb"
            ],
            "Id": "2222222"
        },
        {
            "MatchConfidence": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.12416666746139526,
                    "Top": 0.2968254089355469,
                    "Left": 0.2150000035762787,
                    "Height": 0.23650793731212616
                },
                "Confidence": 99.99958801269531,
                "Pose": {
                    "Yaw": 7.801797866821289,
                    "Roll": -8.326810836791992,
                    "Pitch": 7.844768047332764
                },
                "Quality": {
                    "Sharpness": 86.93206024169922,
                    "Brightness": 79.81291198730469
                },
                "Landmarks": [
                    {
                        "Y": 0.4027804136276245,
                        "X": 0.2575301229953766,
                        "Type": "eyeLeft"
                    },
                    {
```

```
                    "Y": 0.3934555947780609,
                    "X": 0.2956969439983368,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.4309830069541931,
                    "X": 0.2837020754814148,
                    "Type": "nose"
                },
                {
                    "Y": 0.48186683654785156,
                    "X": 0.26812544465065,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.47338807582855225,
                    "X": 0.29905644059181213,
                    "Type": "mouthRight"
                }
            ]
        },
        "Name": "Celeb C",
        "Urls": [
            "www.imdb.com/name/ccccccccc"
        ],
        "Id": "3333333"
    },
    {
        "MatchConfidence": 97.0,
        "Face": {
            "BoundingBox": {
                "Width": 0.11916666477918625,
                "Top": 0.3698412775993347,
                "Left": 0.008333333767950535,
                "Height": 0.22698412835597992
            },
            "Confidence": 99.99999237060547,
            "Pose": {
                "Yaw": 16.38478660583496,
                "Roll": -1.0260354280471802,
                "Pitch": 5.975185394287109
            },
            "Quality": {
                "Sharpness": 83.23492431640625,
```

```
                    "Brightness": 61.408443450927734
                },
                "Landmarks": [
                    {
                        "Y": 0.4632347822189331,
                        "X": 0.049406956881284714,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.46388113498687744,
                        "X": 0.08722897619009018,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.5020678639411926,
                        "X": 0.0758260041475296,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.544157862663269,
                        "X": 0.054029736667871475,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.5463630557060242,
                        "X": 0.08464983850717545,
                        "Type": "mouthRight"
                    }
                ]
            },
            "Name": "Celeb D",
            "Urls": [
                "www.imdb.com/name/ddddddddd"
            ],
            "Id": "4444444"
        }
    ]
}
```


如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的辨識影像中的名人。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考RecognizeCelebrities中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesRequest;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Celebrity;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class RecognizeCelebrities {
    public static void main(String[] args) {
        final String usage = """
                Usage:    <sourceImage>

                Where:
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
```

```
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Locating celebrities in " + sourceImage);
        recognizeAllCelebrities(rekClient, sourceImage);
        rekClient.close();
    }

    public static void recognizeAllCelebrities(RekognitionClient rekClient,
 String sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            RecognizeCelebritiesRequest request =
 RecognizeCelebritiesRequest.builder()
                    .image(souImage)
                    .build();

            RecognizeCelebritiesResponse result =
 rekClient.recognizeCelebrities(request);
            List<Celebrity> celebs = result.celebrityFaces();
            System.out.println(celebs.size() + " celebrity(s) were recognized.
\n");
            for (Celebrity celebrity : celebs) {
                System.out.println("Celebrity recognized: " + celebrity.name());
                System.out.println("Celebrity ID: " + celebrity.id());

                System.out.println("Further information (if available):");
                for (String url : celebrity.urls()) {
                    System.out.println(url);
```

```
                }
                System.out.println();
            }
            System.out.println(result.unrecognizedFaces().size() + " face(s) were
 unrecognized.");

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考RecognizeCelebrities中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun recognizeAllCelebrities(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        RecognizeCelebritiesRequest {
            image = souImage
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.recognizeCelebrities(request)
        response.celebrityFaces?.forEach { celebrity ->
            println("Celebrity recognized: ${celebrity.name}")
            println("Celebrity ID:${celebrity.id}")
```

```
            println("Further information (if available):")
            celebrity.urls?.forEach { url ->
                println(url)
            }
        }
        println("${response.unrecognizedFaces?.size} face(s) were unrecognized.")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK RecognizeCelebrities中的 Kotlin API 參考。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client
```

```
    def recognize_celebrities(self):
        """
        Detects celebrities in the image.

        :return: A tuple. The first element is the list of celebrities found in
                 the image. The second element is the list of faces that were
                 detected but did not match any known celebrities.
        """
        try:
            response =
self.rekognition_client.recognize_celebrities(Image=self.image)
            celebrities = [
                RekognitionCelebrity(celeb) for celeb in
response["CelebrityFaces"]
            ]
            other_faces = [
                RekognitionFace(face) for face in response["UnrecognizedFaces"]
            ]
            logger.info(
                "Found %s celebrities and %s other faces in %s.",
                len(celebrities),
                len(other_faces),
                self.image_name,
            )
        except ClientError:
            logger.exception("Couldn't detect celebrities in %s.",
self.image_name)
            raise
        else:
            return celebrities, other_faces
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件RecognizeCelebrities中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭SearchFaces配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用SearchFaces。

如需詳細資訊，請參閱搜尋人臉 (人臉 ID)。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to find faces in an image that
/// match the face Id provided in the method request.
/// </summary>
public class SearchFacesMatchingId
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

        var rekognitionClient = new AmazonRekognitionClient();

        // Search collection for faces matching the face id.
        var searchFacesRequest = new SearchFacesRequest
        {
            CollectionId = collectionId,
            FaceId = faceId,
            FaceMatchThreshold = 70F,
            MaxFaces = 2,
        };
```

```
            SearchFacesResponse searchFacesResponse = await
 rekognitionClient.SearchFacesAsync(searchFacesRequest);

            Console.WriteLine("Face matching faceId " + faceId);

            Console.WriteLine("Matche(s): ");
            searchFacesResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId} Similarity:
 {face.Similarity}");
            });
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考SearchFaces中的。

CLI

AWS CLI

搜尋系列中符合臉部 ID 的臉孔。

以下指search-faces令會搜尋集合中符合指定臉部 ID 的臉孔。

```
aws rekognition search-faces \
    --face-id 8d3cfc70-4ba8-4b36-9644-90fba29c2dac \
    --collection-id MyCollection
```

輸出：

```
{
    "SearchedFaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
    "FaceModelVersion": "3.0",
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.48166701197624207,
                    "Top": 0.20999999344348907,
                    "Left": 0.21250000596046448,
```

```
                    "Height": 0.36125001311302185
                },
                "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
                "ExternalImageId": "image1.jpg",
                "Confidence": 99.99949645996094,
                "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
            },
            "Similarity": 99.30997467041016
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            },
            "Similarity": 99.24862670898438
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
                "ExternalImageId": "image3.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
            },
            "Similarity": 99.24862670898438
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5349419713020325,
                    "Top": 0.29124999046325684,
```

```
                    "Left": 0.16389399766921997,
                    "Height": 0.40187498927116394
                },
                "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
                "ExternalImageId": "image9.jpg",
                "Confidence": 99.99979400634766,
                "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
            },
            "Similarity": 96.73158264160156
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5307819843292236,
                    "Top": 0.2862499952316284,
                    "Left": 0.1564060002565384,
                    "Height": 0.3987500071525574
                },
                "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
                "ExternalImageId": "image10.jpg",
                "Confidence": 99.99970245361328,
                "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
            },
            "Similarity": 96.48291015625
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5074880123138428,
                    "Top": 0.3774999976158142,
                    "Left": 0.18302799761295319,
                    "Height": 0.3812499940395355
                },
                "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
                "ExternalImageId": "image6.jpg",
                "Confidence": 99.99930572509766,
                "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
            },
            "Similarity": 96.43287658691406
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5574039816856384,
```

```
                    "Top": 0.37187498807907104,
                    "Left": 0.14559100568294525,
                    "Height": 0.4181250035762787
                },
                "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
                "ExternalImageId": "image5.jpg",
                "Confidence": 99.99960327148438,
                "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
            },
            "Similarity": 95.25305938720703
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5773710012435913,
                    "Top": 0.34437501430511475,
                    "Left": 0.12396000325679779,
                    "Height": 0.4337500035762787
                },
                "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
                "ExternalImageId": "image8.jpg",
                "Confidence": 100.0,
                "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
            },
            "Similarity": 95.22837829589844
        }
    ]
}
```


如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的使用臉部 ID 搜尋臉孔。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考SearchFaces中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingImageCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <sourceImage>

                Where:
                   collectionId - The id of the collection. \s
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s

                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String collectionId = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Searching for a face in a collections");
        searchFaceInCollection(rekClient, collectionId, sourceImage);
        rekClient.close();
    }

    public static void searchFaceInCollection(RekognitionClient rekClient, String
collectionId, String sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(new
File(sourceImage));
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            SearchFacesByImageRequest facesByImageRequest =
SearchFacesByImageRequest.builder()
                    .image(souImage)
                    .maxFaces(10)
                    .faceMatchThreshold(70F)
                    .collectionId(collectionId)
                    .build();

            SearchFacesByImageResponse imageResponse =
rekClient.searchFacesByImage(facesByImageRequest);
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face : faceImageMatches) {
                System.out.println("The similarity level is  " +
face.similarity());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
```

```
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考SearchFaces中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.
```

```
        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def search_faces(self, face_id, threshold, max_faces):
        """
        Searches for faces in the collection that match another face from the
        collection.

        :param face_id: The ID of the face in the collection to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: The list of matching faces found in the collection. This list
does
                 not contain the face specified by `face_id`.
        """
        try:
            response = self.rekognition_client.search_faces(
                CollectionId=self.collection_id,
                FaceId=face_id,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            faces = [RekognitionFace(face["Face"]) for face in
response["FaceMatches"]]
            logger.info(
                "Found %s faces in %s that match %s.",
                len(faces),
                self.collection_id,
                face_id,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                face_id,
```

```
            )
            raise
        else:
            return faces
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件SearchFaces中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭SearchFacesByImage配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用SearchFacesByImage。

如需詳細資訊，請參閱搜尋人臉 (映像)。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to search for images matching those
/// in a collection.
/// </summary>
public class SearchFacesMatchingImage
{
    public static async Task Main()
    {
```

```
            string collectionId = "MyCollection";
            string bucket = "bucket";
            string photo = "input.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            // Get an image object from S3 bucket.
            var image = new Image()
            {
                S3Object = new S3Object()
                {
                    Bucket = bucket,
                    Name = photo,
                },
            };

            var searchFacesByImageRequest = new SearchFacesByImageRequest()
            {
                CollectionId = collectionId,
                Image = image,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesByImageResponse searchFacesByImageResponse = await
rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

            Console.WriteLine("Faces matching largest face in image from " +
photo);
            searchFacesByImageResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
            });
        }
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考SearchFacesByImage中的。

CLI

AWS CLI

搜尋集合中符合影像中最大臉孔的臉孔。

以下指search-faces-by-image令會搜尋集合中與指定影像中最大面相符的面。：

```
aws rekognition search-faces-by-image \
    --image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"ExamplePerson.jpg"}}' \
    --collection-id MyFaceImageCollection

{
    "SearchedFaceBoundingBox": {
        "Width": 0.18562500178813934,
        "Top": 0.1618015021085739,
        "Left": 0.5575000047683716,
        "Height": 0.24770642817020416
    },
    "SearchedFaceConfidence": 99.993408203125,
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            },
            "Similarity": 99.97913360595703
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
```

```
            },
            "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
            "ExternalImageId": "image3.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
        },
        "Similarity": 99.97913360595703
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.41499999165534973,
                "Top": 0.09187500178813934,
                "Left": 0.28083300590515137,
                "Height": 0.3112500011920929
            },
            "FaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
            "ExternalImageId": "image2.jpg",
            "Confidence": 99.99769592285156,
            "ImageId": "a294da46-2cb1-5cc4-9045-61d7ca567662"
        },
        "Similarity": 99.18069458007812
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.48166701197624207,
                "Top": 0.20999999344348907,
                "Left": 0.21250000596046448,
                "Height": 0.36125001311302185
            },
            "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
            "ExternalImageId": "image1.jpg",
            "Confidence": 99.99949645996094,
            "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
        },
        "Similarity": 98.66607666015625
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.5349419713020325,
                "Top": 0.29124999046325684,
                "Left": 0.16389399766921997,
```

```
                    "Height": 0.40187498927116394
                },
                "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
                "ExternalImageId": "image9.jpg",
                "Confidence": 99.99979400634766,
                "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
            },
            "Similarity": 98.24278259277344
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5307819843292236,
                    "Top": 0.2862499952316284,
                    "Left": 0.1564060002565384,
                    "Height": 0.3987500071525574
                },
                "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
                "ExternalImageId": "image10.jpg",
                "Confidence": 99.99970245361328,
                "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
            },
            "Similarity": 98.10665893554688
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5074880123138428,
                    "Top": 0.3774999976158142,
                    "Left": 0.18302799761295319,
                    "Height": 0.3812499940395355
                },
                "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
                "ExternalImageId": "image6.jpg",
                "Confidence": 99.99930572509766,
                "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
            },
            "Similarity": 98.10526275634766
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5574039816856384,
                    "Top": 0.37187498807907104,
```

```
                    "Left": 0.14559100568294525,
                    "Height": 0.4181250035762787
                },
                "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
                "ExternalImageId": "image5.jpg",
                "Confidence": 99.99960327148438,
                "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
            },
            "Similarity": 97.94659423828125
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5773710012435913,
                    "Top": 0.34437501430511475,
                    "Left": 0.12396000325679779,
                    "Height": 0.4337500035762787
                },
                "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
                "ExternalImageId": "image8.jpg",
                "Confidence": 100.0,
                "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
            },
            "Similarity": 97.93476867675781
        }
    ],
    "FaceModelVersion": "3.0"
}
```

如需詳細資訊，請參閱 Amazon Rekognition 開發人員指南中的使用影像搜尋臉孔。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考SearchFacesByImage中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.SearchFacesRequest;
import software.amazon.awssdk.services.rekognition.model.SearchFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingIdCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <sourceImage>

                Where:
                   collectionId - The id of the collection. \s
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Searching for a face in a collections");
        searchFacebyId(rekClient, collectionId, faceId);
```

```
        rekClient.close();
    }

    public static void searchFacebyId(RekognitionClient rekClient, String
 collectionId, String faceId) {
        try {
            SearchFacesRequest searchFacesRequest = SearchFacesRequest.builder()
                    .collectionId(collectionId)
                    .faceId(faceId)
                    .faceMatchThreshold(70F)
                    .maxFaces(2)
                    .build();

            SearchFacesResponse imageResponse =
 rekClient.searchFaces(searchFacesRequest);
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face : faceImageMatches) {
                System.out.println("The similarity level is  " +
 face.similarity());
                System.out.println();
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考SearchFacesByImage中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def search_faces_by_image(self, image, threshold, max_faces):
        """
        Searches for faces in the collection that match the largest face in the
        reference image.

        :param image: The image that contains the reference face to search for.
```

```
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: A tuple. The first element is the face found in the reference
image.
                 The second element is the list of matching faces found in the
                 collection.
        """
        try:
            response = self.rekognition_client.search_faces_by_image(
                CollectionId=self.collection_id,
                Image=image.image,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            image_face = RekognitionFace(
                {
                    "BoundingBox": response["SearchedFaceBoundingBox"],
                    "Confidence": response["SearchedFaceConfidence"],
                }
            )
            collection_faces = [
                RekognitionFace(face["Face"]) for face in response["FaceMatches"]
            ]
            logger.info(
                "Found %s faces in the collection that match the largest "
                "face in %s.",
                len(collection_faces),
                image.image_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                image.image_name,
            )
            raise
        else:
            return image_face, collection_faces
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件SearchFacesByImage中的 Python (博托 3) API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 SDK 的 Amazon Rekognition 案例 AWS

下列程式碼範例說明如何使用軟體開發套件在 Amazon Rekognition 中實作常見案例。 AWS 這些案例會展示如何在 Amazon Rekognition 中呼叫多個函數來完成特定任務。每個案例都包含一個連結 GitHub，您可以在其中找到如何設定和執行程式碼的指示。

範例

- 建立 Amazon Rekognition 集合，並使用開發套件在其中尋找臉孔 AWS
- 使用開發套件使用 Amazon Rekognition 偵測和顯示映像中的元素 AWS
- 使用 Amazon Rekognition 和開發套件偵測影片中的資訊 AWS

建立 Amazon Rekognition 集合，並使用開發套件在其中尋找臉孔 AWS

以下程式碼範例顯示做法：

- 建立 Amazon Rekognition 集合。
- 將映像新增到集合中並偵測其中的人臉。
- 在集合中搜尋符合參考映像的人臉。
- 刪除集合。

如需詳細資訊，請參閱搜尋集合中的人臉。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

建立包裝 Amazon Rekognition 函數的類別。

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError
from rekognition_objects import RekognitionFace
from rekognition_image_detection import RekognitionImage

logger = logging.getLogger(__name__)


class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    @classmethod
    def from_file(cls, image_file_name, rekognition_client, image_name=None):
        """
        Creates a RekognitionImage object from a local file.

        :param image_file_name: The file name of the image. The file is opened
 and its
                                bytes are read.
        :param rekognition_client: A Boto3 Rekognition client.
        :param image_name: The name of the image. If this is not specified, the
                           file name is used as the image name.
```

```
        :return: The RekognitionImage object, initialized with image bytes from
 the
                 file.
        """
        with open(image_file_name, "rb") as img_file:
            image = {"Bytes": img_file.read()}
        name = image_file_name if image_name is None else image_name
        return cls(image, name, rekognition_client)


class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
 API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client


    def create_collection(self, collection_id):
        """
        Creates an empty collection.

        :param collection_id: Text that identifies the collection.
        :return: The newly created collection.
        """
        try:
            response = self.rekognition_client.create_collection(
                CollectionId=collection_id
            )
            response["CollectionId"] = collection_id
            collection = RekognitionCollection(response, self.rekognition_client)
            logger.info("Created collection %s.", collection_id)
        except ClientError:
            logger.exception("Couldn't create collection %s.", collection_id)
            raise
        else:
```

```
            return collection


    def list_collections(self, max_results):
        """
        Lists collections for the current account.

        :param max_results: The maximum number of collections to return.
        :return: The list of collections for the current account.
        """
        try:
            response =
 self.rekognition_client.list_collections(MaxResults=max_results)
            collections = [
                RekognitionCollection({"CollectionId": col_id},
 self.rekognition_client)
                for col_id in response["CollectionIds"]
            ]
        except ClientError:
            logger.exception("Couldn't list collections.")
            raise
        else:
            return collections



class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
```

```
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def to_dict(self):
        """
        Renders parts of the collection data to a dict.

        :return: The collection data as a dict.
        """
        rendering = {
            "collection_id": self.collection_id,
            "collection_arn": self.collection_arn,
            "face_count": self.face_count,
            "created": self.created,
        }
        return rendering


    def describe_collection(self):
        """
        Gets data about the collection from the Amazon Rekognition service.

        :return: The collection rendered as a dict.
        """
        try:
            response = self.rekognition_client.describe_collection(
                CollectionId=self.collection_id
            )
```

```
            # Work around capitalization of Arn vs. ARN
            response["CollectionArn"] = response.get("CollectionARN")
            (
                self.collection_arn,
                self.face_count,
                self.created,
            ) = self._unpack_collection(response)
            logger.info("Got data for collection %s.", self.collection_id)
        except ClientError:
            logger.exception("Couldn't get data for collection %s.",
self.collection_id)
            raise
        else:
            return self.to_dict()


    def delete_collection(self):
        """
        Deletes the collection.
        """
        try:

self.rekognition_client.delete_collection(CollectionId=self.collection_id)
            logger.info("Deleted collection %s.", self.collection_id)
            self.collection_id = None
        except ClientError:
            logger.exception("Couldn't delete collection %s.",
self.collection_id)
            raise


    def index_faces(self, image, max_faces):
        """
        Finds faces in the specified image, indexes them, and stores them in the
        collection.

        :param image: The image to index.
        :param max_faces: The maximum number of faces to index.
        :return: A tuple. The first element is a list of indexed faces.
                 The second element is a list of faces that couldn't be indexed.
        """
        try:
            response = self.rekognition_client.index_faces(
                CollectionId=self.collection_id,
```

```
                Image=image.image,
                ExternalImageId=image.image_name,
                MaxFaces=max_faces,
                DetectionAttributes=["ALL"],
            )
            indexed_faces = [
                RekognitionFace({**face["Face"], **face["FaceDetail"]})
                for face in response["FaceRecords"]
            ]
            unindexed_faces = [
                RekognitionFace(face["FaceDetail"])
                for face in response["UnindexedFaces"]
            ]
            logger.info(
                "Indexed %s faces in %s. Could not index %s faces.",
                len(indexed_faces),
                image.image_name,
                len(unindexed_faces),
            )
        except ClientError:
            logger.exception("Couldn't index faces in image %s.",
image.image_name)
            raise
        else:
            return indexed_faces, unindexed_faces


    def list_faces(self, max_results):
        """
        Lists the faces currently indexed in the collection.

        :param max_results: The maximum number of faces to return.
        :return: The list of faces in the collection.
        """
        try:
            response = self.rekognition_client.list_faces(
                CollectionId=self.collection_id, MaxResults=max_results
            )
            faces = [RekognitionFace(face) for face in response["Faces"]]
            logger.info(
                "Found %s faces in collection %s.", len(faces),
self.collection_id
            )
        except ClientError:
```

```
            logger.exception(
                "Couldn't list faces in collection %s.", self.collection_id
            )
            raise
        else:
            return faces


    def search_faces(self, face_id, threshold, max_faces):
        """
        Searches for faces in the collection that match another face from the
        collection.

        :param face_id: The ID of the face in the collection to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: The list of matching faces found in the collection. This list
does
                 not contain the face specified by `face_id`.
        """
        try:
            response = self.rekognition_client.search_faces(
                CollectionId=self.collection_id,
                FaceId=face_id,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            faces = [RekognitionFace(face["Face"]) for face in
response["FaceMatches"]]
            logger.info(
                "Found %s faces in %s that match %s.",
                len(faces),
                self.collection_id,
                face_id,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                face_id,
            )
            raise
        else:
```

```
            return faces


    def search_faces_by_image(self, image, threshold, max_faces):
        """
        Searches for faces in the collection that match the largest face in the
        reference image.

        :param image: The image that contains the reference face to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: A tuple. The first element is the face found in the reference
image.
                 The second element is the list of matching faces found in the
                 collection.
        """
        try:
            response = self.rekognition_client.search_faces_by_image(
                CollectionId=self.collection_id,
                Image=image.image,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            image_face = RekognitionFace(
                {
                    "BoundingBox": response["SearchedFaceBoundingBox"],
                    "Confidence": response["SearchedFaceConfidence"],
                }
            )
            collection_faces = [
                RekognitionFace(face["Face"]) for face in response["FaceMatches"]
            ]
            logger.info(
                "Found %s faces in the collection that match the largest "
                "face in %s.",
                len(collection_faces),
                image.image_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                image.image_name,
```

```
            )
            raise
        else:
            return image_face, collection_faces


class RekognitionFace:
    """Encapsulates an Amazon Rekognition face."""

    def __init__(self, face, timestamp=None):
        """
        Initializes the face object.

        :param face: Face data, in the format returned by Amazon Rekognition
                     functions.
        :param timestamp: The time when the face was detected, if the face was
                          detected in a video.
        """
        self.bounding_box = face.get("BoundingBox")
        self.confidence = face.get("Confidence")
        self.landmarks = face.get("Landmarks")
        self.pose = face.get("Pose")
        self.quality = face.get("Quality")
        age_range = face.get("AgeRange")
        if age_range is not None:
            self.age_range = (age_range.get("Low"), age_range.get("High"))
        else:
            self.age_range = None
        self.smile = face.get("Smile", {}).get("Value")
        self.eyeglasses = face.get("Eyeglasses", {}).get("Value")
        self.sunglasses = face.get("Sunglasses", {}).get("Value")
        self.gender = face.get("Gender", {}).get("Value", None)
        self.beard = face.get("Beard", {}).get("Value")
        self.mustache = face.get("Mustache", {}).get("Value")
        self.eyes_open = face.get("EyesOpen", {}).get("Value")
        self.mouth_open = face.get("MouthOpen", {}).get("Value")
        self.emotions = [
            emo.get("Type")
            for emo in face.get("Emotions", [])
            if emo.get("Confidence", 0) > 50
        ]
        self.face_id = face.get("FaceId")
        self.image_id = face.get("ImageId")
        self.timestamp = timestamp
```

```
    def to_dict(self):
        """
        Renders some of the face data to a dict.

        :return: A dict that contains the face data.
        """
        rendering = {}
        if self.bounding_box is not None:
            rendering["bounding_box"] = self.bounding_box
        if self.age_range is not None:
            rendering["age"] = f"{self.age_range[0]} - {self.age_range[1]}"
        if self.gender is not None:
            rendering["gender"] = self.gender
        if self.emotions:
            rendering["emotions"] = self.emotions
        if self.face_id is not None:
            rendering["face_id"] = self.face_id
        if self.image_id is not None:
            rendering["image_id"] = self.image_id
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        has = []
        if self.smile:
            has.append("smile")
        if self.eyeglasses:
            has.append("eyeglasses")
        if self.sunglasses:
            has.append("sunglasses")
        if self.beard:
            has.append("beard")
        if self.mustache:
            has.append("mustache")
        if self.eyes_open:
            has.append("open eyes")
        if self.mouth_open:
            has.append("open mouth")
        if has:
            rendering["has"] = has
        return rendering
```

使用包裝函式類別從一組映像建立人臉集合，然後搜尋集合中的人臉。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Rekognition face collection demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    rekognition_client = boto3.client("rekognition")
    images = [
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128316.jpg",
            rekognition_client,
            image_name="sitting",
        ),
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128317.jpg",
            rekognition_client,
            image_name="hopping",
        ),
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128318.jpg",
            rekognition_client,
            image_name="biking",
        ),
    ]

    collection_mgr = RekognitionCollectionManager(rekognition_client)
    collection = collection_mgr.create_collection("doc-example-collection-demo")
    print(f"Created collection {collection.collection_id}:")
    pprint(collection.describe_collection())

    print("Indexing faces from three images:")
    for image in images:
        collection.index_faces(image, 10)
    print("Listing faces in collection:")
    faces = collection.list_faces(10)
    for face in faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    print(
```

```
        f"Searching for faces in the collection that match the first face in the
"
        f"list (Face ID: {faces[0].face_id}."
    )
    found_faces = collection.search_faces(faces[0].face_id, 80, 10)
    print(f"Found {len(found_faces)} matching faces.")
    for face in found_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    print(
        f"Searching for faces in the collection that match the largest face in "
        f"{images[0].image_name}."
    )
    image_face, match_faces = collection.search_faces_by_image(images[0], 80, 10)
    print(f"The largest face in {images[0].image_name} is:")
    pprint(image_face.to_dict())
    print(f"Found {len(match_faces)} matching faces.")
    for face in match_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    collection.delete_collection()
    print("Thanks for watching!")
    print("-" * 88)
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用開發套件使用 Amazon Rekognition 偵測和顯示映像中的元素 AWS

以下程式碼範例顯示做法：

- 使用 Amazon Rekognition 偵測映像中的元素。
- 顯示映像並在偵測到的元素周圍繪製邊界方框。

如需詳細資訊，請參閱顯示邊界方框。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

建立類來包裝 Amazon Rekognition 函數。

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError
import requests

from rekognition_objects import (
    RekognitionFace,
    RekognitionCelebrity,
    RekognitionLabel,
    RekognitionModerationLabel,
    RekognitionText,
    show_bounding_boxes,
    show_polygons,
)

logger = logging.getLogger(__name__)


class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
```

```
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    @classmethod
    def from_file(cls, image_file_name, rekognition_client, image_name=None):
        """
        Creates a RekognitionImage object from a local file.

        :param image_file_name: The file name of the image. The file is opened
and its
                                bytes are read.
        :param rekognition_client: A Boto3 Rekognition client.
        :param image_name: The name of the image. If this is not specified, the
                           file name is used as the image name.
        :return: The RekognitionImage object, initialized with image bytes from
the
                 file.
        """
        with open(image_file_name, "rb") as img_file:
            image = {"Bytes": img_file.read()}
        name = image_file_name if image_name is None else image_name
        return cls(image, name, rekognition_client)


    @classmethod
    def from_bucket(cls, s3_object, rekognition_client):
        """
        Creates a RekognitionImage object from an Amazon S3 object.

        :param s3_object: An Amazon S3 object that identifies the image. The
image
                          is not retrieved until needed for a later call.
        :param rekognition_client: A Boto3 Rekognition client.
        :return: The RekognitionImage object, initialized with Amazon S3 object
data.
        """
        image = {"S3Object": {"Bucket": s3_object.bucket_name, "Name":
s3_object.key}}
        return cls(image, s3_object.key, rekognition_client)
```

```
    def detect_faces(self):
        """
        Detects faces in the image.

        :return: The list of faces found in the image.
        """
        try:
            response = self.rekognition_client.detect_faces(
                Image=self.image, Attributes=["ALL"]
            )
            faces = [RekognitionFace(face) for face in response["FaceDetails"]]
            logger.info("Detected %s faces.", len(faces))
        except ClientError:
            logger.exception("Couldn't detect faces in %s.", self.image_name)
            raise
        else:
            return faces


    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.

        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels
            )
            labels = [RekognitionLabel(label) for label in response["Labels"]]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels


    def recognize_celebrities(self):
        """
        Detects celebrities in the image.
```

```
        :return: A tuple. The first element is the list of celebrities found in
                 the image. The second element is the list of faces that were
                 detected but did not match any known celebrities.
        """
        try:
            response =
self.rekognition_client.recognize_celebrities(Image=self.image)
            celebrities = [
                RekognitionCelebrity(celeb) for celeb in
response["CelebrityFaces"]
            ]
            other_faces = [
                RekognitionFace(face) for face in response["UnrecognizedFaces"]
            ]
            logger.info(
                "Found %s celebrities and %s other faces in %s.",
                len(celebrities),
                len(other_faces),
                self.image_name,
            )
        except ClientError:
            logger.exception("Couldn't detect celebrities in %s.",
self.image_name)
            raise
        else:
            return celebrities, other_faces



    def compare_faces(self, target_image, similarity):
        """
        Compares faces in the image with the largest face in the target image.

        :param target_image: The target image to compare against.
        :param similarity: Faces in the image must have a similarity value
greater
                           than this value to be included in the results.
        :return: A tuple. The first element is the list of faces that match the
                 reference image. The second element is the list of faces that
have
                 a similarity value below the specified threshold.
        """
        try:
```

```
            response = self.rekognition_client.compare_faces(
                SourceImage=self.image,
                TargetImage=target_image.image,
                SimilarityThreshold=similarity,
            )
            matches = [
                RekognitionFace(match["Face"]) for match in
response["FaceMatches"]
            ]
            unmatches = [RekognitionFace(face) for face in
response["UnmatchedFaces"]]
            logger.info(
                "Found %s matched faces and %s unmatched faces.",
                len(matches),
                len(unmatches),
            )
        except ClientError:
            logger.exception(
                "Couldn't match faces from %s to %s.",
                self.image_name,
                target_image.image_name,
            )
            raise
        else:
            return matches, unmatches


    def detect_moderation_labels(self):
        """
        Detects moderation labels in the image. Moderation labels identify
content
        that may be inappropriate for some audiences.

        :return: The list of moderation labels found in the image.
        """
        try:
            response = self.rekognition_client.detect_moderation_labels(
                Image=self.image
            )
            labels = [
                RekognitionModerationLabel(label)
                for label in response["ModerationLabels"]
            ]
            logger.info(
```

```
                "Found %s moderation labels in %s.", len(labels), self.image_name
            )
        except ClientError:
            logger.exception(
                "Couldn't detect moderation labels in %s.", self.image_name
            )
            raise
        else:
            return labels


    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [RekognitionText(text) for text in
 response["TextDetections"]]
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

建立輔助函數來繪製邊界框和多邊形。

```
import io
import logging
from PIL import Image, ImageDraw

logger = logging.getLogger(__name__)


def show_bounding_boxes(image_bytes, box_sets, colors):
    """
    Draws bounding boxes on an image and shows it with the default image viewer.
```

```
    :param image_bytes: The image to draw, as bytes.
    :param box_sets: A list of lists of bounding boxes to draw on the image.
    :param colors: A list of colors to use to draw the bounding boxes.
    """
    image = Image.open(io.BytesIO(image_bytes))
    draw = ImageDraw.Draw(image)
    for boxes, color in zip(box_sets, colors):
        for box in boxes:
            left = image.width * box["Left"]
            top = image.height * box["Top"]
            right = (image.width * box["Width"]) + left
            bottom = (image.height * box["Height"]) + top
            draw.rectangle([left, top, right, bottom], outline=color, width=3)
    image.show()



def show_polygons(image_bytes, polygons, color):
    """
    Draws polygons on an image and shows it with the default image viewer.

    :param image_bytes: The image to draw, as bytes.
    :param polygons: The list of polygons to draw on the image.
    :param color: The color to use to draw the polygons.
    """
    image = Image.open(io.BytesIO(image_bytes))
    draw = ImageDraw.Draw(image)
    for polygon in polygons:
        draw.polygon(
            [
                (image.width * point["X"], image.height * point["Y"])
                for point in polygon
            ],
            outline=color,
        )
    image.show()
```

建立類別以剖析 Amazon Rekognition 傳回的剖析物件。

```
class RekognitionFace:
```

```
    """Encapsulates an Amazon Rekognition face."""

    def __init__(self, face, timestamp=None):
        """
        Initializes the face object.

        :param face: Face data, in the format returned by Amazon Rekognition
                     functions.
        :param timestamp: The time when the face was detected, if the face was
                          detected in a video.
        """
        self.bounding_box = face.get("BoundingBox")
        self.confidence = face.get("Confidence")
        self.landmarks = face.get("Landmarks")
        self.pose = face.get("Pose")
        self.quality = face.get("Quality")
        age_range = face.get("AgeRange")
        if age_range is not None:
            self.age_range = (age_range.get("Low"), age_range.get("High"))
        else:
            self.age_range = None
        self.smile = face.get("Smile", {}).get("Value")
        self.eyeglasses = face.get("Eyeglasses", {}).get("Value")
        self.sunglasses = face.get("Sunglasses", {}).get("Value")
        self.gender = face.get("Gender", {}).get("Value", None)
        self.beard = face.get("Beard", {}).get("Value")
        self.mustache = face.get("Mustache", {}).get("Value")
        self.eyes_open = face.get("EyesOpen", {}).get("Value")
        self.mouth_open = face.get("MouthOpen", {}).get("Value")
        self.emotions = [
            emo.get("Type")
            for emo in face.get("Emotions", [])
            if emo.get("Confidence", 0) > 50
        ]
        self.face_id = face.get("FaceId")
        self.image_id = face.get("ImageId")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the face data to a dict.

        :return: A dict that contains the face data.
        """
```

```
        rendering = {}
        if self.bounding_box is not None:
            rendering["bounding_box"] = self.bounding_box
        if self.age_range is not None:
            rendering["age"] = f"{self.age_range[0]} - {self.age_range[1]}"
        if self.gender is not None:
            rendering["gender"] = self.gender
        if self.emotions:
            rendering["emotions"] = self.emotions
        if self.face_id is not None:
            rendering["face_id"] = self.face_id
        if self.image_id is not None:
            rendering["image_id"] = self.image_id
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        has = []
        if self.smile:
            has.append("smile")
        if self.eyeglasses:
            has.append("eyeglasses")
        if self.sunglasses:
            has.append("sunglasses")
        if self.beard:
            has.append("beard")
        if self.mustache:
            has.append("mustache")
        if self.eyes_open:
            has.append("open eyes")
        if self.mouth_open:
            has.append("open mouth")
        if has:
            rendering["has"] = has
        return rendering



class RekognitionCelebrity:
    """Encapsulates an Amazon Rekognition celebrity."""

    def __init__(self, celebrity, timestamp=None):
        """
        Initializes the celebrity object.
```

```
        :param celebrity: Celebrity data, in the format returned by Amazon
 Rekognition
                          functions.
        :param timestamp: The time when the celebrity was detected, if the
 celebrity
                          was detected in a video.
        """
        self.info_urls = celebrity.get("Urls")
        self.name = celebrity.get("Name")
        self.id = celebrity.get("Id")
        self.face = RekognitionFace(celebrity.get("Face"))
        self.confidence = celebrity.get("MatchConfidence")
        self.bounding_box = celebrity.get("BoundingBox")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the celebrity data to a dict.

        :return: A dict that contains the celebrity data.
        """
        rendering = self.face.to_dict()
        if self.name is not None:
            rendering["name"] = self.name
        if self.info_urls:
            rendering["info URLs"] = self.info_urls
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering



class RekognitionPerson:
    """Encapsulates an Amazon Rekognition person."""

    def __init__(self, person, timestamp=None):
        """
        Initializes the person object.

        :param person: Person data, in the format returned by Amazon Rekognition
                       functions.
        :param timestamp: The time when the person was detected, if the person
                          was detected in a video.
        """
```

```
        self.index = person.get("Index")
        self.bounding_box = person.get("BoundingBox")
        face = person.get("Face")
        self.face = RekognitionFace(face) if face is not None else None
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the person data to a dict.

        :return: A dict that contains the person data.
        """
        rendering = self.face.to_dict() if self.face is not None else {}
        if self.index is not None:
            rendering["index"] = self.index
        if self.bounding_box is not None:
            rendering["bounding_box"] = self.bounding_box
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering



class RekognitionLabel:
    """Encapsulates an Amazon Rekognition label."""

    def __init__(self, label, timestamp=None):
        """
        Initializes the label object.

        :param label: Label data, in the format returned by Amazon Rekognition
                      functions.
        :param timestamp: The time when the label was detected, if the label
                          was detected in a video.
        """
        self.name = label.get("Name")
        self.confidence = label.get("Confidence")
        self.instances = label.get("Instances")
        self.parents = label.get("Parents")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the label data to a dict.
```

```
        :return: A dict that contains the label data.
        """
        rendering = {}
        if self.name is not None:
            rendering["name"] = self.name
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering



class RekognitionModerationLabel:
    """Encapsulates an Amazon Rekognition moderation label."""

    def __init__(self, label, timestamp=None):
        """
        Initializes the moderation label object.

        :param label: Label data, in the format returned by Amazon Rekognition
                      functions.
        :param timestamp: The time when the moderation label was detected, if the
                          label was detected in a video.
        """
        self.name = label.get("Name")
        self.confidence = label.get("Confidence")
        self.parent_name = label.get("ParentName")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the moderation label data to a dict.

        :return: A dict that contains the moderation label data.
        """
        rendering = {}
        if self.name is not None:
            rendering["name"] = self.name
        if self.parent_name is not None:
            rendering["parent_name"] = self.parent_name
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering
```

```
class RekognitionText:
    """Encapsulates an Amazon Rekognition text element."""

    def __init__(self, text_data):
        """
        Initializes the text object.

        :param text_data: Text data, in the format returned by Amazon Rekognition
                          functions.
        """
        self.text = text_data.get("DetectedText")
        self.kind = text_data.get("Type")
        self.id = text_data.get("Id")
        self.parent_id = text_data.get("ParentId")
        self.confidence = text_data.get("Confidence")
        self.geometry = text_data.get("Geometry")

    def to_dict(self):
        """
        Renders some of the text data to a dict.

        :return: A dict that contains the text data.
        """
        rendering = {}
        if self.text is not None:
            rendering["text"] = self.text
        if self.kind is not None:
            rendering["kind"] = self.kind
        if self.geometry is not None:
            rendering["polygon"] = self.geometry.get("Polygon")
        return rendering
```

使用包裝函式類別來偵測映像中的元素，並顯示其邊界方框。在這個例子中使用的圖像可以與說明和更多的代碼一 GitHub 起找到。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Rekognition image detection demo!")
```

```
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    rekognition_client = boto3.client("rekognition")
    street_scene_file_name = ".media/pexels-kaique-rocha-109919.jpg"
    celebrity_file_name = ".media/pexels-pixabay-53370.jpg"
    one_girl_url = "https://dhei5unw3vrsx.cloudfront.net/images/
source3_resized.jpg"
    three_girls_url = "https://dhei5unw3vrsx.cloudfront.net/images/
target3_resized.jpg"
    swimwear_object = boto3.resource("s3").Object(
        "console-sample-images-pdx", "yoga_swimwear.jpg"
    )
    book_file_name = ".media/pexels-christina-morillo-1181671.jpg"

    street_scene_image = RekognitionImage.from_file(
        street_scene_file_name, rekognition_client
    )
    print(f"Detecting faces in {street_scene_image.image_name}...")
    faces = street_scene_image.detect_faces()
    print(f"Found {len(faces)} faces, here are the first three.")
    for face in faces[:3]:
        pprint(face.to_dict())
    show_bounding_boxes(
        street_scene_image.image["Bytes"],
        [[face.bounding_box for face in faces]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    print(f"Detecting labels in {street_scene_image.image_name}...")
    labels = street_scene_image.detect_labels(100)
    print(f"Found {len(labels)} labels.")
    for label in labels:
        pprint(label.to_dict())
    names = []
    box_sets = []
    colors = ["aqua", "red", "white", "blue", "yellow", "green"]
    for label in labels:
        if label.instances:
            names.append(label.name)
            box_sets.append([inst["BoundingBox"] for inst in label.instances])
    print(f"Showing bounding boxes for {names} in {colors[:len(names)]}.")
    show_bounding_boxes(
```

```
        street_scene_image.image["Bytes"], box_sets, colors[: len(names)]
    )
    input("Press Enter to continue.")

    celebrity_image = RekognitionImage.from_file(
        celebrity_file_name, rekognition_client
    )
    print(f"Detecting celebrities in {celebrity_image.image_name}...")
    celebs, others = celebrity_image.recognize_celebrities()
    print(f"Found {len(celebs)} celebrities.")
    for celeb in celebs:
        pprint(celeb.to_dict())
    show_bounding_boxes(
        celebrity_image.image["Bytes"],
        [[celeb.face.bounding_box for celeb in celebs]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    girl_image_response = requests.get(one_girl_url)
    girl_image = RekognitionImage(
        {"Bytes": girl_image_response.content}, "one-girl", rekognition_client
    )
    group_image_response = requests.get(three_girls_url)
    group_image = RekognitionImage(
        {"Bytes": group_image_response.content}, "three-girls",
rekognition_client
    )
    print("Comparing reference face to group of faces...")
    matches, unmatches = girl_image.compare_faces(group_image, 80)
    print(f"Found {len(matches)} face matching the reference face.")
    show_bounding_boxes(
        group_image.image["Bytes"],
        [[match.bounding_box for match in matches]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    swimwear_image = RekognitionImage.from_bucket(swimwear_object,
rekognition_client)
    print(f"Detecting suggestive content in {swimwear_object.key}...")
    labels = swimwear_image.detect_moderation_labels()
    print(f"Found {len(labels)} moderation labels.")
    for label in labels:
```

```
            pprint(label.to_dict())
    input("Press Enter to continue.")

    book_image = RekognitionImage.from_file(book_file_name, rekognition_client)
    print(f"Detecting text in {book_image.image_name}...")
    texts = book_image.detect_text()
    print(f"Found {len(texts)} text instances. Here are the first seven:")
    for text in texts[:7]:
        pprint(text.to_dict())
    show_polygons(
        book_image.image["Bytes"], [text.geometry["Polygon"] for text in texts],
"aqua"
    )

    print("Thanks for watching!")
    print("-" * 88)
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 Amazon Rekognition 和開發套件偵測影片中的資訊 AWS

下列程式碼範例示範如何：

- 啟動 Amazon Rekognition 任務，以偵測影片中的人物、物件和文字等元素。
- 檢查工作狀態，直到工作完成。
- 輸出每個工作偵測到的元素清單。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

從 Amazon S3 儲存貯體中的影片中取得名人結果。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.CelebrityRecognitionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.CelebrityRecognition;
import software.amazon.awssdk.services.rekognition.model.CelebrityDetail;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionResponse;
import java.util.List;

/**
 * To run this code example, ensure that you perform the Prerequisites as stated
 * in the Amazon Rekognition Guide:
 * https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
 *
 * Also, ensure that set up your development environment, including your
 * credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class VideoCelebrityDetection {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

       if (args.length != 4) {
           System.out.println(usage);
           System.exit(1);
       }

       String bucket = args[0];
       String video = args[1];
       String topicArn = args[2];
       String roleArn = args[3];
       Region region = Region.US_EAST_1;
       RekognitionClient rekClient = RekognitionClient.builder()
               .region(region)
               .build();

       NotificationChannel channel = NotificationChannel.builder()
               .snsTopicArn(topicArn)
               .roleArn(roleArn)
               .build();

       startCelebrityDetection(rekClient, channel, bucket, video);
       getCelebrityDetectionResults(rekClient);
       System.out.println("This example is done!");
       rekClient.close();
   }

   public static void startCelebrityDetection(RekognitionClient rekClient,
           NotificationChannel channel,
           String bucket,
           String video) {
       try {
           S3Object s3Obj = S3Object.builder()
                   .bucket(bucket)
```

```
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartCelebrityRecognitionRequest recognitionRequest =
StartCelebrityRecognitionRequest.builder()
                    .jobTag("Celebrities")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartCelebrityRecognitionResponse startCelebrityRecognitionResult =
rekClient
                    .startCelebrityRecognition(recognitionRequest);
            startJobId = startCelebrityRecognitionResult.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getCelebrityDetectionResults(RekognitionClient rekClient)
{

        try {
            String paginationToken = null;
            GetCelebrityRecognitionResponse recognitionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (recognitionResponse != null)
                    paginationToken = recognitionResponse.nextToken();

                GetCelebrityRecognitionRequest recognitionRequest =
GetCelebrityRecognitionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .sortBy(CelebrityRecognitionSortBy.TIMESTAMP)
```

```
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds
                while (!finished) {
                    recognitionResponse =
rekClient.getCelebrityRecognition(recognitionRequest);
                    status = recognitionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                VideoMetadata videoMetaData =
recognitionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<CelebrityRecognition> celebs =
recognitionResponse.celebrities();
                for (CelebrityRecognition celeb : celebs) {
                    long seconds = celeb.timestamp() / 1000;
                    System.out.print("Sec: " + seconds + " ");
                    CelebrityDetail details = celeb.celebrity();
                    System.out.println("Name: " + details.name());
                    System.out.println("Id: " + details.id());
                    System.out.println();
                }

            } while (recognitionResponse.nextToken() != null);
```

```
        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

通過標籤偵測操作，偵測影片中的標籤。

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetect {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of the video (for example, people.mp4).\s
                   queueUrl- The URL of a SQS queue.\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
        String roleArn = args[4];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();
```

```
        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startLabels(rekClient, channel, bucket, video);
        getLabelJob(rekClient, sqs, queueUrl);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    public static void startLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartLabelDetectionRequest labelDetectionRequest =
StartLabelDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .minConfidence(50F)
                    .build();

            StartLabelDetectionResponse labelDetectionResponse =
rekClient.startLabelDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

            boolean ans = true;
            String status = "";
            int yy = 0;
            while (ans) {
```

```
                GetLabelDetectionRequest detectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .maxResults(10)
                        .build();

                GetLabelDetectionResponse result =
rekClient.getLabelDetection(detectionRequest);
                status = result.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    ans = false;
                else
                    System.out.println(yy + " status is: " + status);

                Thread.sleep(1000);
                yy++;
            }

            System.out.println(startJobId + " status is: " + status);

        } catch (RekognitionException | InterruptedException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {
        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
                .queueUrl(queueUrl)
                .build();

        try {
            messages = sqs.receiveMessage(messageRequest).messages();

            if (!messages.isEmpty()) {
                for (Message message : messages) {
                    String notification = message.body();

                    // Get the status and job id from the notification
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
```

```
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .build();

                    String jobId = operationJobId.textValue();
                    if (startJobId.compareTo(jobId) == 0) {
                        System.out.println("Job id: " + operationJobId);
                        System.out.println("Status : " +
operationStatus.toString());

                        if (operationStatus.asText().equals("SUCCEEDED"))
                            getResultsLabels(rekClient);
                        else
                            System.out.println("Video analysis failed");

                        sqs.deleteMessage(deleteMessageRequest);
                    } else {
                        System.out.println("Job received was not job " +
startJobId);
                        sqs.deleteMessage(deleteMessageRequest);
                    }
                }
            }

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }

    // Gets the job results by calling GetLabelDetection
```

```
    private static void getResultsLabels(RekognitionClient rekClient) {

        int maxResults = 10;
        String paginationToken = null;
        GetLabelDetectionResponse labelDetectionResult = null;

        try {
            do {
                if (labelDetectionResult != null)
                    paginationToken = labelDetectionResult.nextToken();

                GetLabelDetectionRequest labelDetectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .sortBy(LabelDetectionSortBy.TIMESTAMP)
                        .maxResults(maxResults)
                        .nextToken(paginationToken)
                        .build();

                labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
                VideoMetadata videoMetaData =
labelDetectionResult.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());

                List<LabelDetection> detectedLabels =
labelDetectionResult.labels();
                for (LabelDetection detectedLabel : detectedLabels) {
                    long seconds = detectedLabel.timestamp();
                    Label label = detectedLabel.label();
                    System.out.println("Millisecond: " + seconds + " ");

                    System.out.println("   Label:" + label.name());
                    System.out.println("   Confidence:" +
detectedLabel.label().confidence().toString());

                    List<Instance> instances = label.instances();
                    System.out.println("   Instances of " + label.name());

                    if (instances.isEmpty()) {
```

```
                        System.out.println("        " + "None");
                    } else {
                        for (Instance instance : instances) {
                            System.out.println("        Confidence: " +
 instance.confidence().toString());
                            System.out.println("        Bounding box: " +
 instance.boundingBox().toString());
                        }
                    }
                    System.out.println("   Parent labels for " + label.name() +
 ":");
                    List<Parent> parents = label.parents();

                    if (parents.isEmpty()) {
                        System.out.println("        None");
                    } else {
                        for (Parent parent : parents) {
                            System.out.println("   " + parent.name());
                        }
                    }
                    System.out.println();
                }
            } while (labelDetectionResult != null &&
 labelDetectionResult.nextToken() != null);

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }
}
```

偵測存放於 Amazon S3 儲存貯體中的人臉。

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
```

```
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetect {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of the video (for example, people.mp4).\s
                   queueUrl- The URL of a SQS queue.\s
```

```
                    topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
        String roleArn = args[4];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startLabels(rekClient, channel, bucket, video);
        getLabelJob(rekClient, sqs, queueUrl);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    public static void startLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
```

```
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartLabelDetectionRequest labelDetectionRequest =
StartLabelDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .minConfidence(50F)
                    .build();

            StartLabelDetectionResponse labelDetectionResponse =
rekClient.startLabelDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

            boolean ans = true;
            String status = "";
            int yy = 0;
            while (ans) {

                GetLabelDetectionRequest detectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .maxResults(10)
                        .build();

                GetLabelDetectionResponse result =
rekClient.getLabelDetection(detectionRequest);
                status = result.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    ans = false;
                else
                    System.out.println(yy + " status is: " + status);

                Thread.sleep(1000);
                yy++;
            }

            System.out.println(startJobId + " status is: " + status);
```

```
        } catch (RekognitionException | InterruptedException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {
        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
                .queueUrl(queueUrl)
                .build();

        try {
            messages = sqs.receiveMessage(messageRequest).messages();

            if (!messages.isEmpty()) {
                for (Message message : messages) {
                    String notification = message.body();

                    // Get the status and job id from the notification
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .build();

                    String jobId = operationJobId.textValue();
                    if (startJobId.compareTo(jobId) == 0) {
                        System.out.println("Job id: " + operationJobId);
                        System.out.println("Status : " +
operationStatus.toString());

                        if (operationStatus.asText().equals("SUCCEEDED"))
```

```
                            getResultsLabels(rekClient);
                        else
                            System.out.println("Video analysis failed");

                        sqs.deleteMessage(deleteMessageRequest);
                    } else {
                        System.out.println("Job received was not job " +
startJobId);
                        sqs.deleteMessage(deleteMessageRequest);
                    }
                }
            }

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }

    // Gets the job results by calling GetLabelDetection
    private static void getResultsLabels(RekognitionClient rekClient) {

        int maxResults = 10;
        String paginationToken = null;
        GetLabelDetectionResponse labelDetectionResult = null;

        try {
            do {
                if (labelDetectionResult != null)
                    paginationToken = labelDetectionResult.nextToken();

                GetLabelDetectionRequest labelDetectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .sortBy(LabelDetectionSortBy.TIMESTAMP)
                        .maxResults(maxResults)
                        .nextToken(paginationToken)
                        .build();
```

```
                labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
                VideoMetadata videoMetaData =
labelDetectionResult.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());

                List<LabelDetection> detectedLabels =
labelDetectionResult.labels();
                for (LabelDetection detectedLabel : detectedLabels) {
                    long seconds = detectedLabel.timestamp();
                    Label label = detectedLabel.label();
                    System.out.println("Millisecond: " + seconds + " ");

                    System.out.println("   Label:" + label.name());
                    System.out.println("   Confidence:" +
detectedLabel.label().confidence().toString());

                    List<Instance> instances = label.instances();
                    System.out.println("   Instances of " + label.name());

                    if (instances.isEmpty()) {
                        System.out.println("        " + "None");
                    } else {
                        for (Instance instance : instances) {
                            System.out.println("        Confidence: " +
instance.confidence().toString());
                            System.out.println("        Bounding box: " +
instance.boundingBox().toString());
                        }
                    }
                    System.out.println("   Parent labels for " + label.name() +
":");
                    List<Parent> parents = label.parents();

                    if (parents.isEmpty()) {
                        System.out.println("        None");
                    } else {
                        for (Parent parent : parents) {
                            System.out.println("   " + parent.name());
                        }
```

```
                    }
                    System.out.println();
                }
            } while (labelDetectionResult != null &&
 labelDetectionResult.nextToken() != null);

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }
}
```

偵測 Amazon S3 儲存貯體中存放影片中的不當或冒犯性內容。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import
 software.amazon.awssdk.services.rekognition.model.ContentModerationDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
```

```
 */
public class VideoDetectInappropriate {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

       if (args.length != 4) {
           System.out.println(usage);
           System.exit(1);
       }

       String bucket = args[0];
       String video = args[1];
       String topicArn = args[2];
       String roleArn = args[3];
       Region region = Region.US_EAST_1;
       RekognitionClient rekClient = RekognitionClient.builder()
               .region(region)
               .build();

       NotificationChannel channel = NotificationChannel.builder()
               .snsTopicArn(topicArn)
               .roleArn(roleArn)
               .build();

       startModerationDetection(rekClient, channel, bucket, video);
       getModResults(rekClient);
       System.out.println("This example is done!");
       rekClient.close();
   }
```

```
   public static void startModerationDetection(RekognitionClient rekClient,
           NotificationChannel channel,
           String bucket,
           String video) {

       try {
           S3Object s3Obj = S3Object.builder()
                   .bucket(bucket)
                   .name(video)
                   .build();

           Video vidOb = Video.builder()
                   .s3Object(s3Obj)
                   .build();

           StartContentModerationRequest modDetectionRequest =
StartContentModerationRequest.builder()
                   .jobTag("Moderation")
                   .notificationChannel(channel)
                   .video(vidOb)
                   .build();

           StartContentModerationResponse startModDetectionResult = rekClient
                   .startContentModeration(modDetectionRequest);
           startJobId = startModDetectionResult.jobId();

       } catch (RekognitionException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
   }

   public static void getModResults(RekognitionClient rekClient) {
       try {
           String paginationToken = null;
           GetContentModerationResponse modDetectionResponse = null;
           boolean finished = false;
           String status;
           int yy = 0;

           do {
               if (modDetectionResponse != null)
                   paginationToken = modDetectionResponse.nextToken();
```

```
                GetContentModerationRequest modRequest =
GetContentModerationRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    modDetectionResponse =
rekClient.getContentModeration(modRequest);
                    status = modDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                VideoMetadata videoMetaData =
modDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<ContentModerationDetection> mods =
modDetectionResponse.moderationLabels();
                for (ContentModerationDetection mod : mods) {
                    long seconds = mod.timestamp() / 1000;
                    System.out.print("Mod label: " + seconds + " ");
                    System.out.println(mod.moderationLabel().toString());
                    System.out.println();
                }
```

```
            } while (modDetectionResponse != null &&
 modDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

偵測 Amazon S3 儲存貯體中存放影片中的技術提示區段和鏡頭偵測區段。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartShotDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartTechnicalCueDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionFilters;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.SegmentDetection;
import software.amazon.awssdk.services.rekognition.model.TechnicalCueSegment;
import software.amazon.awssdk.services.rekognition.model.ShotSegment;
import software.amazon.awssdk.services.rekognition.model.SegmentType;
import software.amazon.awssdk.services.sqs.SqsClient;
import java.util.List;

/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectSegment {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();
```

```
        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startSegmentDetection(rekClient, channel, bucket, video);
        getSegmentResults(rekClient);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    public static void startSegmentDetection(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartShotDetectionFilter cueDetectionFilter =
StartShotDetectionFilter.builder()
                    .minSegmentConfidence(60F)
                    .build();

            StartTechnicalCueDetectionFilter technicalCueDetectionFilter =
StartTechnicalCueDetectionFilter.builder()
                    .minSegmentConfidence(60F)
                    .build();

            StartSegmentDetectionFilters filters =
StartSegmentDetectionFilters.builder()
                    .shotFilter(cueDetectionFilter)
                    .technicalCueFilter(technicalCueDetectionFilter)
                    .build();
```

```
            StartSegmentDetectionRequest segDetectionRequest =
StartSegmentDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .segmentTypes(SegmentType.TECHNICAL_CUE, SegmentType.SHOT)
                    .video(vidOb)
                    .filters(filters)
                    .build();

            StartSegmentDetectionResponse segDetectionResponse =
rekClient.startSegmentDetection(segDetectionRequest);
            startJobId = segDetectionResponse.jobId();

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getSegmentResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetSegmentDetectionResponse segDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (segDetectionResponse != null)
                    paginationToken = segDetectionResponse.nextToken();

                GetSegmentDetectionRequest recognitionRequest =
GetSegmentDetectionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    segDetectionResponse =
rekClient.getSegmentDetection(recognitionRequest);
                    status = segDetectionResponse.jobStatusAsString();
```

```
                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }
                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                List<VideoMetadata> videoMetaData =
segDetectionResponse.videoMetadata();
                for (VideoMetadata metaData : videoMetaData) {
                    System.out.println("Format: " + metaData.format());
                    System.out.println("Codec: " + metaData.codec());
                    System.out.println("Duration: " + metaData.durationMillis());
                    System.out.println("FrameRate: " + metaData.frameRate());
                    System.out.println("Job");
                }

                List<SegmentDetection> detectedSegments =
segDetectionResponse.segments();
                for (SegmentDetection detectedSegment : detectedSegments) {
                    String type = detectedSegment.type().toString();
                    if (type.contains(SegmentType.TECHNICAL_CUE.toString())) {
                        System.out.println("Technical Cue");
                        TechnicalCueSegment segmentCue =
detectedSegment.technicalCueSegment();
                        System.out.println("\tType: " + segmentCue.type());
                        System.out.println("\tConfidence: " +
segmentCue.confidence().toString());
                    }

                    if (type.contains(SegmentType.SHOT.toString())) {
                        System.out.println("Shot");
                        ShotSegment segmentShot = detectedSegment.shotSegment();
                        System.out.println("\tIndex " + segmentShot.index());
                        System.out.println("\tConfidence: " +
segmentShot.confidence().toString());
                    }

                    long seconds = detectedSegment.durationMillis();
```

```
                    System.out.println("\tDuration : " + seconds + "
 milliseconds");
                    System.out.println("\tStart time code: " +
 detectedSegment.startTimecodeSMPTE());
                    System.out.println("\tEnd time code: " +
 detectedSegment.endTimecodeSMPTE());
                    System.out.println("\tDuration time code: " +
 detectedSegment.durationSMPTE());
                    System.out.println();
                }

            } while (segDetectionResponse != null &&
 segDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

偵測 Amazon S3 儲存貯體中存放影片中所存的影片文字。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectText {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();
```

```
        startTextLabels(rekClient, channel, bucket, video);
        getTextResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startTextLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getTextResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetTextDetectionResponse textDetectionResponse = null;
            boolean finished = false;
            String status;
```

```
            int yy = 0;

            do {
                if (textDetectionResponse != null)
                    paginationToken = textDetectionResponse.nextToken();

                GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                    status = textDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                VideoMetadata videoMetaData =
textDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<TextDetectionResult> labels =
textDetectionResponse.textDetections();
                for (TextDetectionResult detectedText : labels) {
```

```
                    System.out.println("Confidence: " +
 detectedText.textDetection().confidence().toString());
                    System.out.println("Id : " +
 detectedText.textDetection().id());
                    System.out.println("Parent Id: " +
 detectedText.textDetection().parentId());
                    System.out.println("Type: " +
 detectedText.textDetection().type());
                    System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                    System.out.println();
                }

            } while (textDetectionResponse != null &&
 textDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

偵測 Amazon S3 儲存貯體中存放影片中所存的影片人物。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.PersonDetection;
import java.util.List;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoPersonDetection {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
```

```
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startPersonLabels(rekClient, channel, bucket, video);
        getPersonDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startPersonLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartPersonTrackingRequest personTrackingRequest =
StartPersonTrackingRequest.builder()
                    .jobTag("DetectingLabels")
                    .video(vidOb)
                    .notificationChannel(channel)
                    .build();

            StartPersonTrackingResponse labelDetectionResponse =
rekClient.startPersonTracking(personTrackingRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getPersonDetectionResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
```

```
            GetPersonTrackingResponse personTrackingResult = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (personTrackingResult != null)
                    paginationToken = personTrackingResult.nextToken();

                GetPersonTrackingRequest recognitionRequest =
GetPersonTrackingRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds
                while (!finished) {

                    personTrackingResult =
rekClient.getPersonTracking(recognitionRequest);
                    status = personTrackingResult.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                VideoMetadata videoMetaData =
personTrackingResult.videoMetadata();

                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
```

```
                System.out.println("Job");

                List<PersonDetection> detectedPersons =
 personTrackingResult.persons();
                for (PersonDetection detectedPerson : detectedPersons) {
                    long seconds = detectedPerson.timestamp() / 1000;
                    System.out.print("Sec: " + seconds + " ");
                    System.out.println("Person Identifier: " +
 detectedPerson.person().index());
                    System.out.println();
                }

            } while (personTrackingResult != null &&
 personTrackingResult.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - GetCelebrity識別
  - GetContent適度
  - GetLabel偵測
  - GetPerson追蹤
  - GetSegment偵測
  - GetText偵測
  - StartCelebrity識別
  - StartContent適度
  - StartLabel偵測
  - StartPerson追蹤
  - StartSegment偵測
  - StartText偵測

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

偵測存放於 Amazon S3 儲存貯體中的人臉。

```
suspend fun startFaceDetection(
    channelVal: NotificationChannel?,
    bucketVal: String,
    videoVal: String,
) {
    val s3Obj =
        S3Object {
            bucket = bucketVal
            name = videoVal
        }
    val vidOb =
        Video {
            s3Object = s3Obj
        }

    val request =
        StartFaceDetectionRequest {
            jobTag = "Faces"
            faceAttributes = FaceAttributes.All
            notificationChannel = channelVal
            video = vidOb
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val startLabelDetectionResult = rekClient.startFaceDetection(request)
        startJobId = startLabelDetectionResult.jobId.toString()
    }
}

suspend fun getFaceResults() {
    var finished = false
```

```
    var status: String
    var yy = 0
    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        var response: GetFaceDetectionResponse? = null

        val recognitionRequest =
            GetFaceDetectionRequest {
                jobId = startJobId
                maxResults = 10
            }

        // Wait until the job succeeds.
        while (!finished) {
            response = rekClient.getFaceDetection(recognitionRequest)
            status = response.jobStatus.toString()
            if (status.compareTo("SUCCEEDED") == 0) {
                finished = true
            } else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }

        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = response?.videoMetadata
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
        println("FrameRate: ${videoMetaData?.frameRate}")

        // Show face information.
        response?.faces?.forEach { face ->
            println("Age: ${face.face?.ageRange}")
            println("Face: ${face.face?.beard}")
            println("Eye glasses: ${face?.face?.eyeglasses}")
            println("Mustache: ${face.face?.mustache}")
            println("Smile: ${face.face?.smile}")
        }
    }
}
```

偵測 Amazon S3 儲存貯體中存放影片中的不當或冒犯性內容。

```
suspend fun startModerationDetection(
    channel: NotificationChannel?,
    bucketVal: String?,
    videoVal: String?,
) {
    val s3Obj =
        S3Object {
            bucket = bucketVal
            name = videoVal
        }
    val vidOb =
        Video {
            s3Object = s3Obj
        }
    val request =
        StartContentModerationRequest {
            jobTag = "Moderation"
            notificationChannel = channel
            video = vidOb
        }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val startModDetectionResult = rekClient.startContentModeration(request)
        startJobId = startModDetectionResult.jobId.toString()
    }
}

suspend fun getModResults() {
    var finished = false
    var status: String
    var yy = 0
    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        var modDetectionResponse: GetContentModerationResponse? = null

        val modRequest =
            GetContentModerationRequest {
                jobId = startJobId
                maxResults = 10
            }

        // Wait until the job succeeds.
        while (!finished) {
```

```
            modDetectionResponse = rekClient.getContentModeration(modRequest)
            status = modDetectionResponse.jobStatus.toString()
            if (status.compareTo("SUCCEEDED") == 0) {
                finished = true
            } else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }

        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = modDetectionResponse?.videoMetadata
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
        println("FrameRate: ${videoMetaData?.frameRate}")

        modDetectionResponse?.moderationLabels?.forEach { mod ->
            val seconds: Long = mod.timestamp / 1000
            print("Mod label: $seconds ")
            println(mod.moderationLabel)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS 適用於 Kotlin 的 SDK API 參考》中的下列主題。
  - GetCelebrity識別
  - GetContent適度
  - GetLabel偵測
  - GetPerson追蹤
  - GetSegment偵測
  - GetText偵測
  - StartCelebrity識別
  - StartContent適度
  - StartLabel偵測
  - StartPerson追蹤
  - StartSegment偵測

- StartText偵測

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 SDK 的 Amazon Rekognition 跨服務範例 AWS

下列範例應用程式使用 AWS 開發套件將 Amazon Rekognition 與其他應用程式結合在一起。 AWS 服務每個範例都包含一個連結 GitHub，您可以在其中找到如何設定和執行應用程式的指示。

範例

- 建立相片資產管理應用程式，讓使用者以標籤管理相片
- 使用開發套件使用 Amazon 重新認知功能偵測影像中的個人防護裝置 AWS
- 使用 AWS SDK 偵測影像中的臉孔
- 使用開發套件使用 Amazon Rekognition 偵測影像中的物件 AWS
- 使用開發套件使用 Amazon Rekognition 偵測影片中的人物和物件 AWS
- 使用 SDK 儲存 EXIF 和其他影像資訊 AWS

建立相片資產管理應用程式，讓使用者以標籤管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

.NET

AWS SDK for .NET

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

適用於 C++ 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

適用於 Java 2.x 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

適用於 JavaScript (v3) 的開發套件

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

適用於 Kotlin 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

適用於 PHP 的開發套件

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

適用於 Rust 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用開發套件使用 Amazon 重新認知功能偵測影像中的個人防護裝置 AWS

以下程式碼範例說明如何建置可使用 Amazon Rekognition 在映像中偵測個人防護裝備 (PPE) 的應用程式。

Java

適用於 Java 2.x 的 SDK

說明如何建立使用個人防護裝備偵測影像的 AWS Lambda 功能。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

適用於 JavaScript (v3) 的開發套件

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中的映像中的個人防護設備 (PPE)。該應用程式將結果儲存到 Amazon DynamoDB 資料表中，並使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 使用 Amazon Rekognition 分析映像中是否具有 PPE。
- 驗證 Amazon SES 的電子郵件地址。
- 以結果更新 DynamoDB 資料表。
- 使用 Amazon SES 傳送電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 偵測影像中的臉孔

以下程式碼範例顯示做法：

- 在 Amazon S3 儲存貯體儲存映像。
- 使用 Amazon Rekognition 偵測面部細節，例如年齡範圍、性別和情感 (例如微笑)。
- 顯示這些詳細資訊。

Rust

適用於 Rust 的 SDK

將映像儲存在 Amazon S3 儲存貯體中，並包含上傳字首，使用 Amazon Rekognition 偵測面部細節，例如年齡範圍、性別和情感 (微笑等)，並顯示這些詳細資訊。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition

- Amazon S3

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用開發套件使用 Amazon Rekognition 偵測影像中的物件 AWS

下列程式碼範例說明如何建置可使用 Amazon Rekognition 按類別偵測映像中物件的應用程式。

.NET

AWS SDK for .NET

說明如何使用 Amazon Rekognition .NET API 建立應用程式，該應用程式可使用 Amazon Rekognition 對 Amazon Simple Storage Service (Amazon S3) 儲存貯體中的映像按類別識別物件。此應用程式可使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES

Java

適用於 Java 2.x 的 SDK

說明如何使用 Amazon Rekognition Java API 建立應用程式，該應用程式可使用 Amazon Rekognition 對 Amazon Simple Storage Service (Amazon S3) 儲存貯體中的映像按類別識別物件。此應用程式可使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3

- Amazon SES

JavaScript

適用於 JavaScript (v3) 的開發套件

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立使用 Amazon Rekognition 的應用程式，在位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中的映像中依類別識別物件。此應用程式可使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 使用 Amazon Rekognition 分析映像中的物件。
- 驗證 Amazon SES 的電子郵件地址。
- 使用 Amazon SES 傳送電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES

Kotlin

適用於 Kotlin 的 SDK

展示如何使用 Amazon Rekognition Kotlin API 建立應用程式，該應用程式使用 Amazon Rekognition 對位於 Amazon Simple Storage Service (Amazon S3) 儲存貯體中的映像按類別識別物件。此應用程式可使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition

- Amazon S3
- Amazon SES

Python

適用於 Python (Boto3) 的 SDK

說明如何使用建立可讓您執行下列作業的 Web 應用程式： AWS SDK for Python (Boto3)

- 將相片上傳到 Amazon Simple Storage Service (Amazon S3) 儲存貯體。
- 使用 Amazon Rekognition 分析和標籤照片。
- 使用 Amazon Simple Email Service (Amazon SES) 傳送映像分析的電子郵件報告。

這個例子包含兩個主要組成部分：一個用 React 構建的網頁，以及一個用 Python 編寫的 REST 服務，它是用燒瓶 REST 構建的。 JavaScript

您可以使用 React 網頁執行以下操作：

- 顯示儲存於 S3 儲存貯體中的映像的清單。
- 將映像從您的電腦上傳至 S3 儲存貯體。
- 顯示識別映像中偵測到的專案的映像和標籤。
- 取得 S3 儲存貯體中所有映像的報告，並傳送報告的電子郵件。

該網頁呼叫 REST 服務。該服務將請求發送到 AWS 來執行下列動作：

- 取得並篩選 S3 儲存貯體中的映像的清單。
- 將相片上傳至 S3 儲存貯體。
- 使用 Amazon Rekognition 分析個別照片，並取得標識照片中偵測到的專案的標籤清單。
- 分析 S3 儲存貯體中的所有相片，然後使用 Amazon SES 傳送報告的電子郵件。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用開發套件使用 Amazon Rekognition 偵測影片中的人物和物件 AWS

下列程式碼範例示範如何使用 Amazon Rekognition 偵測映像中的人物和物件。

Java

適用於 Java 2.x 的 SDK

示範如何使用 Amazon Rekognition Java API 來建立應用程式，以偵測位於 Amazon Simple Storage Service (Amazon S3) 儲存貯體的映像中的人臉和物件。此應用程式可使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

適用於 JavaScript (v3) 的開發套件

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中影片中的臉部和物件。此應用程式可使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 使用 Amazon Rekognition 分析映像中是否具有 PPE。
- 驗證 Amazon SES 的電子郵件地址。
- 使用 Amazon SES 傳送電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3

- Amazon SES

Python

適用於 Python (Boto3) 的 SDK

使用 Amazon Rekognition 透過啟動非同步偵測任務來偵測映像中的人臉、物件和人物。此範例也會設定 Amazon Rekognition 以在任務完成時通知 Amazon Simple Notification Service (Amazon SNS) 主題，並訂閱 Amazon Simple Queue Service (Amazon SQS) 佇列到該主題。當佇列收到有關任務的訊息時，會擷取任務並輸出結果。

此範例最佳檢視於 GitHub。有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon SNS
- Amazon SQS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 SDK 儲存 EXIF 和其他影像資訊 AWS

以下程式碼範例顯示做法：

- 從 JPG、JPEG 或 PNG 檔案中取得 EXIF 資訊。
- 將映像檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Rekognition 識別檔案中的三個主要屬性 (標籤)。
- 將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表。

Rust

適用於 Rust 的 SDK

從 JPG、JPEG 或 PNG 檔案獲取 EXIF 資訊，將映像檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Rekognition 識別三個主要屬性 (Amazon Rekognition 中的標籤)，然後將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表中。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 SDK 使用 Rekognition AWS。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

API 參考

亞馬遜重新認知 API 參考現在位於亞馬遜重新認知 API 參考。

亞馬遜重新認知安全

雲端安全是 AWS 最重視的一環。身為 AWS 的客戶，您將能從資料中心和網路架構中獲益，這些都是專為最重視安全的組織而設計的。

您可以使用下列主題來了解如何保護您的 Amazon Rekognition 資源。

主題

Amazon Rekognition 的身分與存取管理

AWS Identity and Access Management (IAM) 可協助管理員安全地控制 AWS 資源存取權。 AWS 服務 IAM 管理員可控制哪些人員可進行身分驗證 (簽署) 並獲得授權 (具有許可) 以使用 Amazon Rekognition 資源。IAM 是您可以使用的 AWS 服務 ，無需額外付費。

主題

物件

您使用 AWS Identity and Access Management (IAM) 的方式會有所不同，具體取決於您在 Amazon Rekognition 中所做的工作。

服務使用者 ：如果您使用 Amazon Rekognition 服務來執行任務，您的管理員會為您提供必要的憑證和許可。隨著您為了執行作業而使用的 Amazon Rekognition 功能數量變多，您可能會需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 Amazon Rekognition 中的特徵，請參閱 Amazon Rekognition 身分和存取的疑難排解。

服務管理員：若您在公司負責管理 Amazon Rekognition 資源，您應該具備 Amazon Rekognition 的完整存取權限。您的任務是判斷服務使用者應存取的 Amazon Rekognition 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司可搭配 Amazon Rekognition 使用 IAM 的方式，請參閱 Amazon Rekognition 如何與 IAM 搭配運作。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 Amazon Rekognition 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的範例 Amazon Rekognition 身分型政策，請參閱 Amazon Rekognition 身分型政策範例。

使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。 AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。 AWS IAM Identity Center (IAM 身分中心) 使用者、貴公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料都是聯合身分識別的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使 AWS 用同盟存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》 AWS 帳戶中的如何登入您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署您的要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。如需更多資訊，請參閱 AWS IAM Identity Center 使用者指南中的多重要素驗證和 IAM 使用者指南中的在 AWS中使用多重要素驗證 (MFA)。

IAM 使用者和群組

IAM 使用者是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。如需進一步了解，請參閱IAM 使用者指南中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以切換角色，在中暫時擔任 IAM 角色。 AWS Management Console 您可以透過呼叫 AWS CLI 或 AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱 IAM 使用者指南中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱 IAM 使用者指南中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源（而不是使用角色作為代理）。若要了解跨帳戶存取角色和以資源為基礎的政策之間的差異，請參閱 IAM 使用者指南中的 IAM 中的跨帳戶資源存取。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。

- 轉寄存取工作階段 (FAS) — 當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
- 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱 IAM 使用者指南中的建立角色以委派許可給 AWS 服務服務。
- 服務連結角色 — 服務連結角色是連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

- 在 Amazon EC2 上執行的應用程式 — 您可以使用 IAM 角色來管理在 EC2 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並提供給其所有應用程式，請建立連接至執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需詳細資訊，請參閱 IAM 使用者指南中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱 IAM 使用者指南中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS ，當與身分識別或資源相關聯時，會定義其權限。 AWS 當主參與者 (使用者、root 使用者或角色工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以 JSON 文件的形式儲存在中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該原則的使用者可以從 AWS Management Console AWS CLI、或 AWS API 取得角色資訊。

使用身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶. 受管政策包括 AWS 受管政策和客戶管理的策略。如需了解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

使用以資源為基礎的政策

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 和 Amazon VPC 是支援 ACL 的服務範例。 AWS WAF如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的原則類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交

集。會在 Principal 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的 IAM 實體許可界限。

- 服務控制策略（SCP）— SCP 是 JSON 策略，用於指定中組織或組織單位（OU）的最大權限。AWS Organizations AWS Organizations 是一種用於分組和集中管理您企業擁 AWS 帳戶 有的多個服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需 Organizations 和 SCP 的詳細資訊，請參閱 AWS Organizations 使用者指南中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。要了解如何在涉及多個政策類型時 AWS 確定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

Amazon Rekognition 如何與 IAM 搭配運作

在您使用 IAM 管理對 Amazon Rekognition 的存取權之前，您應該瞭解哪些 IAM 功能可以與 Amazon Rekognition 搭配使用。若要取得 Amazon Rekognition 和其他 AWS 服務如何與 IAM 搭配運作的高階檢視，請參閱 IAM 使用者指南中的與 IAM 搭配使用的AWS 服務。

主題

- Amazon Rekognition 身分型政策
- Amazon Rekognition 資源型政策
- Amazon Rekognition IAM 角色

Amazon Rekognition 身分型政策

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。Amazon Rekognition 支援特定動作、資源和條件鍵。若要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 JSON 政策元素參考。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼条件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。原則動作通常與關聯的 AWS API 作業具有相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

Amazon Rekognition 中的政策動作會在動作之前使用下列前綴：`rekognition:`。例如，若要授予某人使用 Amazon Rekognition `DetectLabels` API 操作的許可，來偵測影像中的物件、場景或概念，請在其政策中加入 `rekognition:DetectLabels` 動作。政策陳述式必須包含 `Action` 或 `NotAction` 元素。Amazon Rekognition 會定義自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": [
      "rekognition:action1",
      "rekognition:action2"
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 `Describe` 文字的所有動作，請包含以下動作：

```
"Action": "rekognition:Describe*"
```

若要查看 Amazon Rekognition 動作清單，請參閱《IAM 使用者指南》中的 Amazon Rekognition 定義的動作。

資源

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。陳述式必須包含 `Resource` 或 `NotResource` 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

如需 ARN 格式的詳細資訊，請參閱 Amazon 資源名稱 (ARN) 和 AWS 服務命名空間。

例如，若要在陳述式中指定 MyCollection 集合，請使用以下 ARN。

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/MyCollection"
```

若要指定屬於特定帳戶的所有執行個體，請使用萬用字元 (*)：

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/*"
```

有些 Amazon Rekognition 動作無法對特定資源執行，例如用來建立資源的動作。在這些情況下，您必須使用萬用字元 (*)。

```
"Resource": "*"
```

若要查看 Amazon Rekognition 資源類型及其 ARN 的清單，請參閱《IAM 使用者指南》中的 Amazon Rekognition 定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 Amazon Rekognition 定義的動作。

條件索引鍵

Amazon Rekognition 不提供任何服務專用條件索引鍵，但它支援一些全域條件索引鍵的使用。若要查看所有 AWS 全域條件金鑰，請參閱 IAM 使用者指南中的AWS 全域條件內容金鑰。

Amazon Rekognition 資源型政策

僅在複製自訂標籤模型時，Amazon Rekognition 才支援以資源為基礎的政策。如需詳細資訊，請參閱 Amazon Rekognition 資源型政策範例。

其他服務 (例如 Amazon S3) 也支援以資源為基礎的許可政策。例如，您可以將政策連接至 S3 儲存貯體，以管理該儲存貯體的存取許可。

若要存取存放在 Amazon S3 儲存貯體中的影像，您必須具備存取 S3 儲存貯體中物件的許可。若有此許可，Amazon Rekognition 就可以從 S3 儲存貯體下載影像。下列政策範例可讓使用者在名為 Tests3bucket 的 S3 儲存貯體上執行 s3:GetObject 動作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::Tests3bucket/*"
            ]
        }
    ]
}
```

若要使用啟用版本控制的 S3 儲存貯體，請新增 `s3:GetObjectVersion` 動作，如下列範例所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::Tests3bucket/*"
            ]
        }
    ]
```

Amazon Rekognition IAM 角色

IAM 角色是您 AWS 帳戶中具有特定許可的實體。

將暫時憑證與 Amazon Rekognition 搭配使用

您可以搭配聯合使用暫時憑證、擔任 IAM 角色，或是擔任跨帳戶角色。您可以透過呼叫AssumeRole或GetFederation權杖等 AWS STS API 作業來取得臨時安全登入資料。

Amazon Rekognition 支援使用暫時憑證。

服務連結角色

服務連結角色可讓 AWS 服務存取其他服務中的資源，以代表您完成動作。服務連結角色會顯示在您的 IAM 帳戶中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

Amazon Rekognition 不支援服務連結角色。

服務角色

此功能可讓服務代表您擔任服務角色。此角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶中，且由該帳戶所擁有。這表示 IAM 管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

Amazon Rekognition 支援服務角色。

使用服務角色可能會造成安全問題，其中 Amazon Rekognition 會用來呼叫其他服務，並對其無法存取的資源採取行動。為了確保您的帳戶安全，您應該限制 Amazon Rekognition 只能存取正在使用的資源的範圍。這可以通過將信任政策附加到您的 IAM 服務角色來完成。如需如何執行此作業的資訊，請參閱預防跨服務混淆代理人。

在 Amazon Rekognition中選擇 IAM 角色

當您設定 Amazon Rekognition 來分析儲存影片時，必須選擇角色，以允許 Amazon Rekognition 代您存取 Amazon SNS。如果您之前已建立服務角色或服務連結角色，Amazon Rekognition 會提供您角色清單讓您選擇。如需詳細資訊，請參閱 the section called “設定 Amazon Rekognition Video”。

AWS Amazon Rekognition 的受管政策

若要新增使用者、群組和角色的權限，使用 AWS 受管理的原則比自己撰寫原則更容易。建立 IAM 客戶受管政策需要時間和專業知識，而受管政策可為您的團隊提供其所需的許可。若要快速開始使用，您可以使用我們的 AWS 受管政策。這些政策涵蓋常見使用案例，並可在您的 AWS 帳戶中使用。如需 AWS 受管政策的詳細資訊，請參閱 IAM 使用者指南中的AWS 受管政策。

AWS 服務會維護和更新 AWS 受管理的策略。您無法變更 AWS 受管理原則中的權限。服務偶爾會在 AWS 受管政策中新增其他許可以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群組和角色)。當新功能啟動或新操作可用時，服務很可能會更新 AWS 受管政策。服務不會從 AWS 受管理的政策移除權限，因此政策更新不會破壞您現有的權限。

此外，還 AWS 支援跨多個服務之工作職能的受管理原則。例如，ReadOnly存取 AWS 管理原則會提供所有 AWS 服務和資源的唯讀存取權。當服務啟動新功能時，會為新作業和資源新 AWS 增唯讀權限。如需任務職能政策的清單和說明，請參閱 IAM 使用者指南中有關任務職能的AWS 受管政策。

AWS 受管政策： AmazonRekognitionFullAccess

AmazonRekognitionFullAccess 授予 Amazon Rekognition 資源的完整存取，包括建立與刪除集合。

您可將 AmazonRekognitionFullAccess 政策連接到 IAM 身分。

許可詳細資訊

此政策包含以下許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 受管政策： AmazonRekognitionReadOnlyAccess

AmazonRekognitionReadOnlyAccess 授予 Amazon Rekognition 資源的唯獨存取。

您可將 AmazonRekognitionReadOnlyAccess 政策連接到 IAM 身分。

許可詳細資訊

此政策包含以下許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AmazonRekognitionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
                "rekognition:SearchFacesByImage",
                "rekognition:DetectText",
                "rekognition:GetCelebrityInfo",
                "rekognition:RecognizeCelebrities",
                "rekognition:DetectModerationLabels",
                "rekognition:GetLabelDetection",
                "rekognition:GetFaceDetection",
                "rekognition:GetContentModeration",
                "rekognition:GetPersonTracking",
                "rekognition:GetCelebrityRecognition",
                "rekognition:GetFaceSearch",
                "rekognition:GetTextDetection",
                "rekognition:GetSegmentDetection",
                "rekognition:DescribeStreamProcessor",
                "rekognition:ListStreamProcessors",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DetectProtectiveEquipment",
                "rekognition:ListTagsForResource",
                "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset",
                "rekognition:ListProjectPolicies",
                "rekognition:ListUsers",
                "rekognition:SearchUsers",
                "rekognition:SearchUsersByImage",
                "rekognition:GetMediaAnalysisJob",
                "rekognition:ListMediaAnalysisJobs"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

AWS 受管政策： AmazonRekognitionServiceRole

AmazonRekognitionServiceRole 可讓 Amazon Rekognition 代表您呼叫 Amazon Kinesis Data Streams 和 Amazon SNS 服務。

您可將 AmazonRekognitionServiceRole 政策連接到 IAM 身分。

如果使用此服務角色，您應該透過限制 Amazon Rekognition 只能存取您正在使用的資源的範圍，以確保帳戶安全。這可以通過將信任政策附加到您的 IAM 服務角色來完成。如需如何執行此作業的資訊，請參閱 預防跨服務混淆代理人。

許可詳細資訊

此政策包含以下許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:AmazonRekognition*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "arn:aws:kinesis:*:*:stream/AmazonRekognition*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

AWS 受管政策： AmazonRekognitionCustomLabelsFullAccess

此政策適用於 Amazon Rekognition 自訂標籤；使用者。使用此 AmazonRekognitionCustomLabelsFullAccess 政策可讓使用者完整存取 Amazon Rekognition 自訂標籤 API，以及完整存取 Amazon Rekognition 自訂標籤主控台所建立的主控台儲存貯體。

許可詳細資訊

此政策包含以下許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:ListAllMyBuckets",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*custom-labels*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CopyProjectVersion",
                "rekognition:CreateProject",
                "rekognition:CreateProjectVersion",
                "rekognition:StartProjectVersion",
                "rekognition:StopProjectVersion",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DeleteProject",
```

```
                "rekognition:DeleteProjectVersion"
                "rekognition:TagResource",
                "rekognition:UntagResource",
                "rekognition:ListTagsForResource",
                "rekognition:CreateDataset",
                "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset",
                "rekognition:UpdateDatasetEntries",
                "rekognition:DistributeDatasetEntries",
                "rekognition:DeleteDataset",
                "rekognition:PutProjectPolicy",
                "rekognition:ListProjectPolicies",
                "rekognition:DeleteProjectPolicy"

            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Rekognition 更新受管政策 AWS

檢視有關 Amazon Rekognition AWS 受管政策更新的詳細資訊，因為此服務開始追蹤這些變更。如需有關此頁面變更的自動提醒，請訂閱 Amazon Rekognition 文件历史記錄頁面上的 RSS 摘要。

變更	描述	日期
涉及媒體分析工作的動作已新增至下列受管理政策： • AWS 受管政策：AmazonRekognitionReadOnlyAccess	Amazon Rekognition 將下列動作新增至 AmazonRekognitionReadOnlyAccess 受管政策： • GetMediaAnalysisJob • ListMediaAnalysisJob	2023 年 10 月 31 日
與管理使用者有關的動作已新增至下列受管理政策：	Amazon Rekognition 將下列動作新增至 AmazonRek	2023 年 6 月 12 日

變更	描述	日期
• AWS 受管政策：AmazonRekognitionReadOnlyAccess	ognitionReadOnlyAccess 受管政策： • ListUsers • SearchUsers • SearchUsersByImage	
已將「自訂標籤模型複製」的動作 ProjectPolicy 和「自訂標籤模型複製」新增至下列受管 • AWS 受管政策：AmazonRekognitionFullAccess • AWS 受管政策：AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition 在 AmazonRekognitionCustomLabelsFullAccess 和 AmazonRekognitionFullAccess 受管政策中新增了下列動作： • CopyProjectVersion • PutProjectPolicy • ListProjectPolicies • DeleteProjectPolicy	2022 年 7 月 21 日
已將「自訂標籤模型複製」的動作 ProjectPolicy 和「自訂標籤模型複製」新增至下列受管 • AWS 受管政策：AmazonRekognitionReadOnlyAccess	Amazon Rekognition 將下列動作新增至受管政策： AmazonRekognitionReadOnlyAccess • ListProjectPolicies	2022 年 7 月 21 日

變更	描述	日期
下列受管政策的資料集管理更新： • AWS 受管政策：AmazonRekognitionReadOnlyAccess • AWS 受管政策：AmazonRekognitionFullAccess • AWS 受管政策：AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition 將下列動作新增到 AmazonRekognitionReadOnlyAccess AmazonRekognitionFullOnlyAccess、和受管政策 AmazonRekognitionCustomLabelsFullAccess • `CreateDataset` • `ListDatasetEntries` • `ListDatasetLabels` • `DescribeDataset` • `UpdateDatasetEntries` • `DistributeDatasetEntries` • `DeleteDataset`	2021 年 11 月 1 日
AWS 受管政策：AmazonRekognitionReadOnlyAccess 和 AWS 受管政策：AmazonRekognitionFullAccess 的標籤更新	Amazon Rekognition 在 AmazonRekognitionFullAccess 和政策中添加了新的標記操作。 AmazonRekognitionReadOnlyAccess	2021 年 4 月 2 日
Amazon Rekognition 開始追蹤變更	Amazon Rekognition 開始追蹤其 AWS 受管政策的變更。	2021 年 4 月 2 日

Amazon Rekognition 身分型政策範例

根據預設，使用者和角色不具備建立或修改 Amazon Rekognition 資源的許可。他們也無法使用 AWS Management Console AWS CLI、或 AWS API 執行工作。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 操作的所需許可。管理員接著必須將這些政策連接至需要這些許可的使用者或群組。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱 IAM 使用者指南中的在 JSON 索引標籤上建立政策。

主題

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 Amazon Rekognition 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管原則並邁向最低權限權限 — 若要開始授與使用者和工作負載的權限，請使用可授與許多常見使用案例權限的AWS 受管理原則。它們在您的 AWS 帳戶. 建議您透過定義特定於您使用案例的 AWS 客戶管理政策，進一步降低使用權限。如需更多資訊，請參閱 IAM 使用者指南中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。您也可以使用條件來授與對服務動作的存取權 (如透過特定) 使用這些動作 AWS 服務，例如 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您編寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 IAM Access Analyzer 政策驗證。
- 需要多因素身份驗證 (MFA) — 如果您的案例需要 IAM 使用者或根使用者 AWS 帳戶，請開啟 MFA 以獲得額外的安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。

使用 Amazon Rekognition 主控台

除了 Amazon Rekognition 自訂標籤特徵之外，使用 Amazon Rekognition 主控台時，Amazon Rekognition 不需要任何額外許可。如需有關 Amazon Rekognition 自訂標籤的詳細資訊，請參閱步驟 5：設定 Amazon Rekognition 自訂標籤主控台許可。

您不需要為僅對 AWS CLI 或 AWS API 進行呼叫的使用者允許最低主控台權限。反之，只需允許存取符合您嘗試執行之 API 操作的動作即可。

Amazon Rekognition 自訂標籤政策範例

您可以為 Amazon Rekognition 自訂標籤建立身分類型政策。如需詳細資訊，請參閱安全性。

範例 1：允許使用者唯讀存取資源

下列範例授予對 Amazon Rekognition 資源的唯讀存取權。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
                "rekognition:SearchFacesByImage",
                "rekognition:DetectText",
                "rekognition:GetCelebrityInfo",
                "rekognition:RecognizeCelebrities",
                "rekognition:DetectModerationLabels",
                "rekognition:GetLabelDetection",
                "rekognition:GetFaceDetection",
                "rekognition:GetContentModeration",
                "rekognition:GetPersonTracking",
                "rekognition:GetCelebrityRecognition",
                "rekognition:GetFaceSearch",
                "rekognition:GetTextDetection",
```

```
                "rekognition:GetSegmentDetection",
                "rekognition:DescribeStreamProcessor",
                "rekognition:ListStreamProcessors",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DetectProtectiveEquipment",
                "rekognition:ListTagsForResource",
               "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset"

            ],
            "Resource": "*"
        }
    ]
}
```

範例 2：允許使用者完整存取資源

下列範例授予對 Amazon Rekognition 資源的完整存取權。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:*"
            ],
            "Resource": "*"
        }
    ]
}
```

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此原則包含在主控台上或以程式設計方式使用 AWS CLI 或 AWS API 完成此動作的權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Rekognition 資源型政策範例

Amazon Rekognition 自訂標籤使用以資源為基礎的政策 (稱為專案政策) 來管理模型版本的複製許可。

專案原則會授與或拒絕將模型版本從來源專案複製到目標專案的權限。如果目標專案位於不同的帳戶中，或者您想要限制 AWS 帳戶內的存取權，則需要專案政策。例如，您可能想要拒絕特定 IAM 角色的複製許可。 AWS 如需詳細資訊，請參閱複製模型。

准許複製模型版本

下列範例可讓主體 `arn:aws:iam::123456789012:role/Admin` 複製模型版本 `arn:aws:rekognition:us-east-1:123456789012:project/my_project/version/test_1/1627045542080`。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Principal":{
        "AWS":"arn:aws:iam::123456789012:role/Admin"
      },
      "Action":"rekognition:CopyProjectVersion",
      "Resource":"arn:aws:rekognition:us-east-1:123456789012:project/my_project/
version/test_1/1627045542080"
    }
  ]
}
```

Amazon Rekognition 身分和存取的疑難排解

請使用以下資訊來協助您診斷和修正使用 Amazon Rekognition 和 IAM 時可能遇到的常見問題。

主題

- 我未獲授權，不得在 Amazon Rekognition 中執行動作
- 我沒有授權執行 iam：PassRole
- 我是管理員，想要允許其他人存取 Amazon Rekognition
- 我想要允許我 AWS 帳戶以外的人員存取我的 Amazon Rekognition 資源

我未獲授權，不得在 Amazon Rekognition 中執行動作

如果 AWS Management Console 告訴您您沒有執行動作的授權，則您必須聯絡您的管理員以尋求協助。您的管理員是為您提供簽署憑證的人員。

以下範例錯誤會在 `mateojackson` IAM 使用者嘗試使用主控台檢視 *`widget`* 的詳細資訊，但卻沒有 `rekognition:`*`GetWidget`* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rekognition:GetWidget on resource: my-example-widget
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *`my-example-widget`* 動作存取 `rekognition:`*`GetWidget`* 資源。

我沒有授權執行 iam：PassRole

如果您收到錯誤，告知您無權執行 `iam:PassRole` 動作，您的政策必須更新，允許您將角色傳遞給 Amazon Rekognition。

有些 AWS 服務 允許您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 `marymajor` 的 IAM 使用者嘗試使用主控台在 Amazon Rekognition 中執行動作時，發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 系統管理員。您的管理員提供您的簽署憑證。

我是管理員，想要允許其他人存取 Amazon Rekognition

若要允許其他人存取 Amazon Rekognition，您必須針對需要存取的人員或應用程式建立 IAM 實體 (使用者或角色)。他們將使用該實體的憑證來存取 AWS。您接著必須將政策連線到實體，在 Amazon Rekognition 中授予正確的許可。

若要立即開始使用，請參閱《IAM 使用者指南》中的建立您的第一個 IAM 委派使用者及群組。

我想要允許我 AWS 帳戶以外的人員存取我的 Amazon Rekognition 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 Amazon Rekognition 是否支援這些功能，請參閱 Amazon Rekognition 如何與 IAM 搭配運作。
- 若要了解如何提供對您所擁有資源 AWS 帳戶 的存取權，請參閱《IAM 使用者指南》中您擁有的另一 AWS 帳戶 個 IAM 使用者提供存取權限。
- 若要了解如何將資源存取權提供給第三方 AWS 帳戶，請參閱 IAM 使用者指南中的提供第三方 AWS 帳戶 擁有的存取權。

- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 若要了解跨帳戶存取使用角色和以資源為基礎的政策之間的差異，請參閱 IAM 使用者指南中的 IAM 中的跨帳戶資源存取。

Amazon Rekognition 的資料保護

AWS 共同責任模型適用於 Amazon Rekognition 中的資料保護。如此模型所述，AWS 負責保護執行所有 AWS 雲端 的全球基礎設施。您負責維護在此基礎設施上託管內容的控制權。您也必須負責您所使用 AWS 服務 的安全組態和管理任務。如需有關資料隱私權的更多相關資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 AWS 共同的責任模型和 GDPR 部落格文章。

基於資料保護目的，建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 憑證，並設定個人使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 AWS CloudTrail 設定 API 和使用者活動日誌記錄。
- 使用 AWS 加密解決方案，以及 AWS 服務 內的所有預設安全控制項。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在透過命令列介面或 API 存取 AWS 時，需要 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如 Name (名稱) 欄位。其包括您在使用主控台、API、AWS CLI 或 AWS SDK 時操作 Rekognition 或其他 AWS 服務 。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

資料加密

下列資訊說明 Amazon Rekognition 使用資料加密來保護您的資料的情況。

靜態加密

Amazon Rekognition Image

映像

除非您透過造訪 AI 服務退出政策頁面並遵循此處說明的程序來選擇退出，否則可儲存傳送至 Amazon Rekognition API 操作的映像並用於改善服務。儲存的映像會在靜態時使用 AWS Key Management Service (Amazon S3) 進行加密 (SSE-KMS)。

集合

對於在集合中存放資訊的臉部比較操作，偵測演算法將先偵測輸入映像中的臉部，擷取至每個臉部的特徵向量中，接著將臉部向量存放於集合。Amazon Rekognition 在執行臉部比較時會使用這些臉部向量。臉部向量儲存為浮點數的陣列。

Amazon Rekognition Video

影片

若要分析影片，Amazon Rekognition 會將影片複製到服務中進行處理。除非您透過造訪 AI 服務退出政策頁面並遵循此處說明的程序來選擇退出，否則影片可能會儲存並用於改善服務。這些影片會在靜態時使用 AWS Key Management Service (SSE-KMS) 進行加密 (Amazon S3)。

Amazon Rekognition自訂標籤

Amazon Rekognition 自訂標籤會加密靜態資料。

映像

為了訓練您的模型，Amazon Rekognition 自訂標籤會製作一份來源訓練和測試映像的複本。複製的映像會在 Amazon 簡單儲存服務 (S3) 中使用伺服器端加密，與您提供的 AWS KMS key 或 AWS 擁有的 KMS 金鑰進行靜態加密。Amazon Rekognition 自訂標籤只支援對稱 KMS 金鑰。您的來源映像不受影響。如需詳細資訊，請參閱什麼是 Amazon Rekognition 自訂標籤。

模型

根據預設，Amazon Rekognition 自訂標籤會搭配 AWS 擁有的金鑰 使用伺服器端加密保護資料將儲存在 Amazon S3 儲存貯體的受訓模型和清單檔案加密。如需詳細資訊，請參閱使用伺服器端加密保護資料。訓練結果會寫入至`OutputConfig`輸入參數中指定的值區CreateProjectVersion。訓練結果會使用儲存貯體 (`OutputConfig`) 所設定的加密加密組態。

主控台儲存貯體

Amazon Rekognition 自訂標籤主控台會建立可用來管理專案的 Amazon S3 儲存貯體 (主控台儲存貯體)。主控台儲存貯體使用預設的 Amazon S3 加密進行加密。如需詳細資訊，請參閱 Amazon Simple Storage Service 的 S3 儲存貯體預設加密。如果您使用自有 KMS 金鑰，請在建立主控台儲存貯體之後進行設定。如需詳細資訊，請參閱使用伺服器端加密保護資料。Amazon Rekognition 自訂標籤會封鎖對主控台儲存貯體的公開存取。

Rekognition Face Liveness

儲存在 Rekognition Face Liveness 服務帳戶中的所有工作階段相關資料都會完全靜態加密。依預設，參考和稽核映像會使用服務帳戶中的AWS擁有金鑰加密。不過，您可以選擇提供自有 AWS KMS 金鑰來加密這些映像。

傳輸中加密

Amazon Rekognition API 端點僅支援透過 HTTPS 的安全連線。所有通訊都是使用 Transport Layer Security (TLS) 加密。

金鑰管理

您可以使用 AWS Key Management Service (KMS) 來管理存放在 Amazon S3 儲存貯體中輸入映像和影片的金鑰。如需詳細資訊，請參閱 AWS Key Management Service 概念。

臉部活體的客戶受管金鑰加密

該 CreateFaceLivenessSessionAPI 採用一個可選`KmsKeyId`參數。您可以提供帳戶中建立 KMS 的 `id` 金鑰。此密鑰將用於加密 StartFaceLivenessSessionAPI 期間獲得的參考和審核圖像，並且在 GetFaceLivenessSessionResultsAPI 期間，將使用此密鑰解密圖像，然後再返回結果。如果 CreateFaceLivenessSession 請求包含 OutputConfig，參考和稽核映像將上傳到指定的 Amazon S3 路徑。我們建議您在 Amazon S3 儲存貯體中啟用伺服器端加密 (SSE-S3)，以便資料在靜態時繼續保持加密狀態。

當您提供自有 AWS KMS 金鑰 ID 時，Rekognition Face Liveness 服務會取得許可，呼叫 API 的主體使用客戶受管金鑰。用於從客戶後端 (API `CreateFaceLivenessSession` 和 `GetFaceLivenessSessionResults`) 調用 API 的主體 (使用者或角色) 必須具有存取權，才能執行下列動作：

- 公里：DescribeKey
- 公里：GenerateDataKey

- kms:解密

網際網路流量隱私權

適用於 Amazon Rekognition 的 Amazon Virtual Private Cloud (Amazon VPC) 端點是 VPC 中的邏輯實體，僅允許連線到 Amazon Rekognition。Amazon VPC 會將請求路由至 Amazon Rekognition，並將回應路由回 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 端點。如需將 Amazon VPC 端點的相關資訊和 Amazon Rekognition 搭配使用，請參閱 將亞馬遜重新認知與 Amazon VPC 端點搭配使用。

將亞馬遜重新認知與 Amazon VPC 端點搭配使用

如果您使用亞馬遜虛擬私有雲 (Amazon VPC) 託管 AWS 資源，則可以在 VPC 和亞馬遜 Rekognition 之間建立私有連接。您可以使用此連線來啟用 Amazon Rekognition 不用透過公有網際網路在 VPC 與您的資源進行通訊。

Amazon VPC 是一種 AWS 服務，可用來在您定義的虛擬網路中啟動 AWS 資源。您可利用 VPC 來控制您的網路設定，例如 IP 地址範圍、子網路、路由表和網路閘道。AWS 網路使用 VPC 端點處理 VPC 與 AWS 服務之間的路由。

若要將您的 VPC 連接到亞馬遜認知，您需要為亞馬遜網站定義一個介面 VPC 端點。界面端點是包含私有 IP 地址的彈性網路界面，做為通往支援之 AWS 服務的流量進入點。端點為 Amazon Rekognition 提供可靠、可擴展的連線能力，不需要網際網路閘道、網路位址轉譯 (NAT) 執行個體或 VPN 連線。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的什麼是 Amazon VPC。

AWS 已啟用介面虛擬私人雲端端點PrivateLink。此 AWS 技術讓私有通訊使用於彈性網路介面與 AWS 服務之間的私有 IP 地址。

> **Note**
>
> AWS 支援所有亞馬遜重新認知聯邦資訊處理標準 (FIPS) 端點PrivateLink。

為亞馬遜重新建立亞馬遜 VPC 端點

您可以建立兩種類型的 Amazon VPC 端點，以搭配使用亞馬遜重新認知。

- 與 Amazon 重新認知作業搭配使用的 VPC 端點。這是適用於大部份使用者的最合適 VPC 端點類型。

- 適用於 Amazon Rekognition 作業的 VPC 端點，其端點符合聯邦資訊處理標準 (FIPS) 出版物 140-2 美國政府標準。

若要開始在您的 VPC 上使用亞馬遜重新認知，請使用 Amazon VPC 主控台建立適用於 Amazon Rekognition 的界面 VPC 端點。如需說明，請參閱建立界面端點中的「使用主控台建立 AWS 服務的界面端點」。請記下下列步驟：

- 步驟 3 — 對於服務類別，選擇AWS 服務。
- 步驟 4 — 對於服務名稱，選擇下列其中一個選項：
  - 亞馬遜. 区域. 雷克尼— 為 Amazon 重新認知作業建立 VPC 端點。
  - 亞馬遜. 區域. 雷克認知-菲普斯— 使用符合聯邦資訊處理標準 (FIPS) 出版物 140-2 美國政府標準的端點，為 Amazon Rekognition 作業建立 VPC 端點。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的入門。

為亞馬遜重新建立 VPC 端點政策

您可以為亞馬遜虛擬私人雲端節點建立政策，以指定下列項目：

- 可執行動作的主體。
- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱 Amazon VPC 使用者指南中的使用 VPC 端點控制服務的存取。

下列範例政策可讓使用者透過 VPC 端點連線至 Amazon Rekognition 來呼叫`DetectFaces`API 操作。該政策可防止使用者透過虛擬私人雲端端點執行其他 Amazon Rekognition API 操作。

使用者仍然可以從 VPC 外部呼叫其他 Amazon Rekognition API 作業。如需如何拒絕存取 VPC 以外的 Amazon Rekognition API 作業的相關資訊，請參閱Amazon Rekognition 身分型政策。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "rekognition:DetectFaces"
```

```
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Principal": "*"
        }
    ]
}
```

若要修改亞馬遜重新認知的 VPC 端點政策

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 如果您尚未為 Amazon 建立端點，請選擇建立端點。然後選取 com.amazonaws.***Region***.rekognition，然後選擇 Create endpoint (建立端點)。
3. 在導覽窗格中選擇 Endpoints (端點)。
4. 選取 com.amazonaws.***Region***.rekognition 端點，然後選擇螢幕下半部的 Policy (政策) 標籤。
5. 選擇 Edit Policy (編輯政策)，並對政策做出變更。

亞馬遜重新認知的合規驗證

第三方稽核員會評估 Amazon Rekognition 的安全性和合規性，作為多個稽核人員的一部分AWS合規方案。這些計劃包括 SOC、PCI、FedRAMP、HIPAA 等等。

如需特定合規計劃範圍內的 AWS 服務清單，請參閱合規計劃內的 AWS 服務。如需一般資訊，請參閱 AWS 合規計劃。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱在 AWS Artifact 中下載報告。

您在使用 Amazon Rekognition 時的合規責任取決於資料的敏感度、公司的合規目標以及適用的法律和法規。AWS提供下列資源以協助遵循法規：

- 安全與合規快速入門指南 – 這些部署指南討論在 AWS 上部署以安全及合規為重心基準環境的架構考量和步驟。
- HIPAA 安全與合規架構白皮書 – 本白皮書說明公司可如何運用 AWS 來建立 HIPAA 合規的應用程式。
- AWS 合規資源 – 這組手冊和指南可能適用於您的產業和位置。
- AWS Config – 此 AWS 服務可評定資源組態與內部實務、業界準則和法規的合規狀態。

- AWS Security Hub – 此 AWS 服務可供您檢視 AWS 中的安全狀態，可助您檢查是否符合安全產業標準和最佳實務。

亞馬遜重新認知中的韌性

AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。AWS區域提供多個分開且隔離的實際可用區域，並以低延遲、高輸送量和高度備援聯網功能相互連結。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需有關 AWS 區域與可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

除了AWS全球基礎設施 Amazon Rekognition 提供多種功能，協助支援您的資料彈性和備份需求。

亞馬遜重新認知中的組態和漏洞分析

組態和 IT 控制是 AWS 與身為我們客戶的您共同的責任。如需詳細資訊，請參閱 AWS 共同的責任模型。

預防跨服務混淆代理人

在AWS，當一個服務時，可能會發生跨服務模擬 (呼叫服務) 呼叫其他服務 (稱為服務). 呼叫服務可以被操縱以對其他客戶的資源採取行動，即使它不應該具有適當的權限，從而導致混淆的副問題。

為了預防這種情況，AWS 提供的工具可協助您保護所有服務的資料，而這些服務主體已獲得您帳戶中資源的存取權。

我們建議您使用aws:SourceArn和aws:SourceAccount資源政策中的全域條件內容金鑰，可限制 Amazon Rekognition 為資源提供其他服務的許可。

如果值aws:SourceArn不包含帳戶識別碼 (例如 Amazon S3 儲存貯體 ARN)，您必須使用這兩個金鑰來限制許可。如果您同時使用鍵和aws:SourceArn值包含帳戶 ID，aws:SourceAccount價值和帳戶aws:SourceArn在同一個保單聲明中使用時，價值必須使用相同的帳戶 ID。

如果您想要僅允許一個資源與跨服務存取相關聯，則請使用 aws:SourceArn。如果您想要允許該帳戶中的任何資源與跨服務使用相關聯，請使用 aws:SourceAccount。

的價值aws:SourceArn必須是 Rekognition 所使用之資源的 ARN，並以下列格式指定：arn:aws:rekognition:region:account:resource。

的價值arn:User ARN應該是將呼叫視訊分析作業的使用者 (擔任角色的使用者) 的 ARN。

混淆副問題的推薦方法是使用aws:SourceArn具有完整資源 ARN 的全局條件上下文鍵。

如果您不知道資源的完整 ARN，或者您要指定多個資源，請使用aws:SourceArn含萬用字元的金鑰 (*) 對於 ARN 的未知部分。例如：arn:aws:*rekognition*:*:111122223333:*。

為了防止混淆的副問題，執行以下步驟：

1. 在 IAM 主控台的導覽窗格中，選擇角色選項。主控台會顯示您目前帳戶的角色。
2. 選擇您要修改的角色名稱。您修改的角色應具有AmazonRekognitionServiceRole權限原則。選擇信任關係標籤。
3. 選擇 Edit trust policy (編輯信任政策)。
4. 在「」編輯信任原則」頁面上，以使用其中一個或兩者的政策取代預設 JSON 政策aws:SourceArn和aws:SourceAccount全域條件上下文索引鍵。請參閱下列範例原則。
5. 選擇 Update policy (更新政策)。

下列範例是信任原則，說明如何使用aws:SourceArn和aws:SourceAccount亞馬遜 Rekognition 中的全局條件上下文鍵，以防止混淆的副問題。

如果您正在使用儲存和串流影片，您可以在 IAM 角色中使用類似下列的政策：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"rekognition.amazonaws.com",
            "AWS":"arn:User ARN"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            },
            "StringLike":{

 "aws:SourceArn":"arn:aws:rekognition:region:111122223333:streamprocessor/*"
            }
         }
```

```
      }
   ]
}
```

如果您只使用儲存的影片，您可以在 IAM 角色中使用類似下列的政策 (請注意，您不需要包含StringLike引數，指定streamprocessor):

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"rekognition.amazonaws.com",
            "AWS":"arn:User ARN"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            }
         }
      }
   ]
}
```

亞馬遜重新認知中的基礎設施安全

作為一項受管服務，亞馬遜 Rekognition 受到保護AWS全球網絡安全。如需有關 AWS 安全服務以及 AWS 如何保護基礎設施的詳細資訊，請參閱 AWS 雲端安全。若要使用基礎設施安全性的最佳實務來設計您的 AWS 環境，請參閱安全性支柱 AWS 架構良好的框架中的基礎設施保護。

您使用AWS透過網路存取亞馬遜重新認知的已發佈 API 呼叫。用戶端必須支援下列項目：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密 (PFS) 的密碼套件，例如 DHE (Ephemeral Diffie-Hellman) 或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。

此外，請求必須使用存取索引鍵 ID 和與 IAM 主體相關聯的私密存取索引鍵來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

監控 Amazon Rekognition

監控是維護 Amazon Rekognition 及其他 AWS 解決方案的可靠性、可用性和效能所不可或缺。AWS 提供以下監控工具可觀看 Rekognition、在發生錯誤時回報以及在適當情況下自動採取行動：

- 亞馬遜 CloudWatch監控您的AWS資源及其他應用程式AWS實時。您可以收集和追蹤指標、建立自訂儀表板，以及設定警示，在特定指標達到您指定的閾值時通知您或採取動作。例如，你可以 CloudWatch 追蹤 Amazon EC2 執行個體的 CPU 使用率或其他指標，並在需要時自動啟動新執行個體。如需詳細資訊，請參閱亞馬遜 CloudWatch 使用者指南。
- 亞馬遜 CloudWatch 日誌可讓您監控、存放和存取 Amazon EC2 執行個體的日誌檔。 CloudTrail，以及其他來源。 CloudWatch 記錄可監控記錄檔中的資訊，並在符合特定臨界值的情況下通知您。您也可以將日誌資料存檔在高耐用性的儲存空間。如需詳細資訊，請參閱亞馬遜 CloudWatch 日誌使用者指南。
- 亞馬遜 EventBridge可用於自動化您的AWS服務並自動回應應用程式可用性問題或資源變更等系統事件。活動來自AWS服務交付給 EventBridge 在接近實時的情況。您可編寫簡單的規則，來指示您在意的事件，以及當事件符合規則時所要自動執行的動作。如需詳細資訊，請參閱亞馬遜 EventBridge 使用者指南。
- AWS CloudTrail 擷取您 AWS 帳戶發出或代表發出的 API 呼叫和相關事件，並傳送記錄檔案至您指定的 Simple Storage Service (Amazon S3) 儲存貯體。您可以找出哪些使用者和帳戶呼叫 AWS、發出呼叫的來源 IP 地址，以及呼叫的發生時間。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南。

使用亞馬遜監控 Rekognition 要手段 CloudWatch

同 CloudWatch，您可以取得個別 Rekognition 作業的指標或帳戶的全域 Rekognition 指標，您可以使用指標追蹤 Rekognition 解決方案的健康狀態，並設定警示以在一或多個指標落在定義的閾值之外時通知您。例如，您可以查看發生的伺服器錯誤次數指標，或者偵測到的臉孔數量指標。您也可以查看特定 Rekognition 作業成功次數的量度。要查看指標，您可以使用亞馬遜 CloudWatch,亞馬遜AWS Command Line Interface，或CloudWatch API。

您也可以使用 Rekognition 主控台查看所選期間內的彙總量度。如需詳細資訊，請參閱練習 4：查看彙總指標 (主控台)。

使用 CloudWatch Rekognition 度量

要使用指標，您必須指定下列資訊：

- 指標維度，或者無維度。維度是一組用來單獨辨識指標的名稱值組。Rekognition 具有一個維度，已命名操作。它提供特定操作的指標。如果您未指定維度，則量度的範圍會限制在您帳戶內的所有 Rekognition 作業。
- 指標名稱，例如 `UserErrorCount`。

您可以使用以下方式取得 Rekognition 的監視資料AWS Management Console，該AWS CLI，或 CloudWatch API。您也可以 CloudWatch 透過其中一個 Amazon AWS 軟體開發套件 (SDK) 或 CloudWatch API 工具。控制台顯示基於原始數據的一系列圖形 CloudWatch API。根據需求，您可能偏好使用顯示於主控台中的圖形或自 API 擷取的圖形。

下列清單顯示一些常見的指標用途。這些是協助您開始的建議，而不是完整清單。

我要如何？	相關指標
我要如何追蹤辨識到的臉孔數量？	監控 `DetectedFaceCount` 指標的 `Sum` 數據。
我要如何得知我的應用程式已達每秒最高請求數量？	監控 `ThrottledCount` 指標的 `Sum` 數據。
我要如何監控請求錯誤？	使用 `UserErrorCount` 指標的 `Sum` 統計資料。
我要如何找到請求總數？	使用 `ResponseTime` 指標的 `ResponseTime` 與 `Data Samples` 統計資料。其中包括任何產生錯誤的請求。如果您只想要查看成功操作呼叫，請使用 `SuccessfulRequestCount` 指標。
我要如何監控 `Rekognition` 操作呼叫的延遲？	使用 `ResponseTime` 指標。
我怎樣才能監視多少次`IndexFaces` 已成功將臉孔加入至 Rekognition 集合？	使用 `SuccessfulRequestCount` 指標與 `IndexFaces` 操作來監控 `Sum` 統計資料。使用 `Operation` 維度來選擇操作與指標。

你必須有適當的 CloudWatch 使用以監視 Rekognition 的權限 CloudWatch。(如需詳細資訊，請參閱 Amazon CloudWatch 的身分驗證和存取控制)。

存取 Rekognition 手段

下列範例說明如何使用 CloudWatch 控制台，AWS CLI，以及 CloudWatchAPI。

檢視指標 (主控台)

1. 開啟手段 CloudWatch 控制台https://console.aws.amazon.com/cloudwatch/。
2. 選擇 Metrics (指標)、選擇 All Metrics (所有指標) 標籤，然後選擇 Rekognition。
3. 選擇 Metrics with no dimensions (無維度的指標)，然後選擇一個指標。

   例如，選擇DetectedFace測量已偵測到多少張臉孔的度量。
4. 選擇日期範圍的值。指標計數顯示於圖形中。

若要查看一段時間內的成功指標 **`DetectFaces`** 操作呼叫 (CLI)。

- 開啟 AWS CLI，然後輸入下列命令：

  ```
  aws cloudwatch get-metric-statistics --metric-name
  SuccessfulRequestCount --start-time 2017-1-1T19:46:20 --end-time
  2017-1-6T19:46:57 --period 3600 --namespace AWS/Rekognition --
  statistics Sum --dimensions Name=Operation,Value=DetectFaces --region
  us-west-2
  ```

  此範例顯示在一段時間內執行的成功 `DetectFaces` 操作呼叫。如需詳細資訊，請參閱get-metric-statistics。

若要存取量度 (CloudWatch API)

- 呼叫 `GetMetricStatistics`。如需詳細資訊，請參閱亞馬遜 CloudWatch API 參考資料。

建立警示

您可以建立 CloudWatch 警示狀態變更時，會傳送 Simple Notification Service (Amazon SNS) 訊息的警示。警示會監看指定時段內的單一指標，並根據與多個時段內指定閾值相對的指標值來執行一或多個動作。此動作是傳送到 Amazon SNS 主題或 Auto Scaling 政策的通知。

警示僅會針對持續狀態變更呼叫動作。 CloudWatch 警報不會僅僅因為它們處於特定狀態所不可或缺。狀態必須發生變更並維持一段指定的時間。

若要設定警示 (主控台)

1. 登入AWS Management Console並打開 CloudWatch 控制台https://console.aws.amazon.com/cloudwatch/。
2. 選擇 Create Alarm (建立警示)。這會啟動 Create Alarm Wizard (建立警示精靈)。
3. 在 Metrics with no dimensions (無維度的指標) 指標清單中，選擇 Rekognition Metrics (Rekognition 指標)，然後選擇一個指標。

   例如，選擇DetectedFaceCount以設定偵測到的臉孔數目上限的警示。
4. 在 Time Range (時間範圍) 區域內，選擇包含您已呼叫的臉部偵測操作之日期範圍值。選擇 Next (下一步)
5. 填入 Name (名稱) 和 Description (說明)。對於 Whenever (每當) 選項，請選擇 >= 並輸入您所選擇的最大值。
6. 如果你想 CloudWatch 當達到警報狀態時，向您發送電子郵件，每當這個警報：，選擇狀態為「警報」。若要將警示傳送至現有的 Amazon SNS 主題，傳送通知至：中，選擇現有的 SNS 主題。若要設定新電子郵件訂閱清單的名稱和電子郵件地址，請選擇建立主題 CloudWatch 保存列表並將其顯示在字段中，以便您可以使用它來設置將 future 的警報。

   > **Note**
   >
   > 如果您使用建立主題若要建立新 Amazon SNS 主題，必須先驗證電子郵件地址，預定收件者才能收到通知。Amazon SNS 只會在警示進入警示狀態時傳送電子郵件。如果此警示狀態在驗證電子郵件地址之前發生變更，目標收件人就不會收到通知。
7. 在 Alarm Preview (警示預覽) 區段中預覽警示。選擇 Create Alarm (建立警示)。

設定警示 (AWS CLI)

- 開啟 AWS CLI，然後輸入下列命令。變更的值`alarm-actions`參數，以參照您先前建立的 Amazon SNS 主題。

  ```
  aws cloudwatch put-metric-alarm --alarm-name UserErrors --
  alarm-description "Alarm when more than 10 user errors occur"
  --metric-name UserErrorCount --namespace AWS/Rekognition --
  ```

```
statistic Average --period 300 --threshold 10 --comparison-
operator GreaterThanThreshold --evaluation-periods 2 --alarm-actions
arn:aws:sns:us-west-2:111111111111:UserError --unit Count
```

此範例說明如何建立警示，當 5 分鐘內發生超過 10 個使用者錯誤時通知。如需詳細資訊，請參閱put-metric-alarm。

若要設定鬧鐘 (CloudWatch API)

- 呼叫 `PutMetricAlarm`。如需詳細資訊，請參閱亞馬遜 CloudWatch API 參考資料。

CloudWatchRekognition 度量

本節包含亞馬遜的相關資訊 CloudWatch 度量和操作維度可用於 Amazon Rekognition。

您也可以從 Rekognition 主控台查看 Rekognition 量度的彙總檢視。如需詳細資訊，請參閱練習 4：查看彙總指標 (主控台)。

CloudWatch Rekognition 度量

下表摘要「Rekognition 要手段」測量結果。

指標	描述
SuccessfulRequestCount	成功請求的數量。成功請求的回應碼範圍是 200 到 299。 單位：計數 有效的統計資訊：`Sum,Average`
ThrottledCount	已調節的請求數目。Rekognition 會在收到的要求數量超過您帳戶設定的每秒交易限制時，將會調節要求。如果經常超過為您的帳戶所設的限制，您可以請求提高上限。若要請求提高，請參閱 AWS Service Limits。 單位：計數 有效的統計資訊：`Sum,Average`
ResponseTime	Rekognition 計算回應的時間 (以毫秒為單位)。

指標	描述
	單位： 1. Data Samples 統計資料的計數 2. Average 統計資料的毫秒 有效的統計資訊：Data Samples,Average Note 該ResponseTime 「Rekognition」度量窗格中不包含量度。
DetectedFaceCount	以 IndexFaces 或 DetectFaces 操作偵測到臉部的次數。 單位：計數 有效的統計資訊：Sum,Average
DetectedLabelCount	以 DetectLabels 操作偵測到標籤的次數。 單位：計數 有效的統計資訊：Sum,Average
ServerErrorCount	伺服器錯誤的次數。伺服器錯誤的回應碼範圍是 500 到 599。 單位：計數 有效的統計資訊：Sum, Average
UserErrorCount	使用者錯誤次數 (無效參數、無效影像、無權限等)。使用者錯誤的回應碼範圍是 400 到 499。 單位：計數 有效的統計資訊：Sum,Average

指標	描述
MinInferenceUnits	期間所指定的推論單位數下限。StartProjectVersion 請求。 單位：計數 有效的統計資訊：Average
MaxInferenceUnits	期間所指定的推論單元數上限。StartProjectVersion 請求。 單位：計數 有效的統計資訊：Average
DesiredInferenceUnits	Rekognition 要向上或縮減的推論單位數目。 單位：計數 有效的統計資訊：Average
InServiceInferenceUnits	模型正在使用的推論單位數。 單位：計數 有效的統計資訊：Average 建議您使用「平均」統計資料來取得使用多少執行處理的 1 分鐘平均值。

CloudWatch Rekognition 串流的量度

Rekognition 也有用於串流作業的第二個命名空間，即「Rekognition 串流」。下表摘要「重要手段串流」測量結果。

指標	描述
SuccessfulRequestCount	成功請求的數量。成功請求的回應碼範圍是 200 到 299。 單位：計數 有效的統計資訊：Sum,Average

指標	描述
CallCount	在您的帳戶中執行的指定操作數目。 有效的統計資訊：Sum,Average
ThrottledCount	已調節的請求數目。Rekognition 會在收到的要求數量超過您帳戶設定的每秒交易限制時，將會調節要求。如果經常超過為您的帳戶所設的限制，您可以請求提高上限。若要請求提高，請參閱 AWS Service Limits。 單位：計數 有效的統計資訊：Sum,Average
ServerErrorCount	伺服器錯誤的次數。伺服器錯誤的回應碼範圍是 500 到 599。 單位：計數 有效的統計資訊：Sum, Average
UserErrorCount	使用者錯誤次數 (無效參數、無效影像、無權限等)。使用者錯誤的回應碼範圍是 400 到 499。 單位：計數 有效的統計資訊：Sum,Average

CloudWatch 重要手段

若要擷取特定操作的指標，請使用 Rekognition 命名空間並提供操作維度。

如需維度的更多資訊，請參閱尺寸在亞馬遜 CloudWatch 使用者指南。

CloudWatch Rekognition 自訂標籤的維度

下表顯示 CloudWatch 可與 Rekognition 自訂標籤搭配使用的維度：

維度	描述
ProjectName	您使用建立的 Rekognition 自訂標籤專案的名稱 `CreateProject` 。
VersionName	您使用建立的 Rekognition 自訂標籤專案版本名稱`CreateProjectVersion` 。

如需維度的更多資訊，請參閱尺寸在亞馬遜 CloudWatch 使用者指南。

使用記錄亞馬遜重新認知 API 呼叫AWS CloudTrail

亞馬遜重新認知與AWS CloudTrail，提供使用者、角色或使用者所採取之動作記錄的服務AWS亞馬遜重新認知服務。CloudTrail將亞馬遜重新認知的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 Amazon Rekognition 主控台的呼叫，以及對 Amazon Rekognition API 操作的程式碼呼叫。如果您建立追蹤，您可以啟用持續傳遞CloudTrail亞馬遜 S3 儲存貯體的事件，包括亞馬遜重新認知的事件。如果您不設定追蹤記錄，仍然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新的事件。使用所收集的資訊CloudTrail，您可以決定向 Amazon Rekognition 發出的請求、提出請求的 IP 位址、提出請求的人員、提出要求的時間以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱 AWS CloudTrail 使用者指南。

亞馬遜重新認知資訊CloudTrail

當您建立帳戶時，系統會在您的 AWS 帳戶中啟用 CloudTrail。當活動發生在亞馬遜 Rekognition 中時，該活動會記錄在CloudTrail與其他一起事件AWS服務事件事件歷史。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷程記錄檢視事件。

在您的事件的持續記錄AWS帳戶，包括亞馬遜 Rekognition 的事件，創建一個跟踪。一個線索啟用 CloudTrail將日誌文件交付到亞馬遜 S3 存儲桶。根據預設，當您在主控台建立追蹤記錄時，追蹤記錄會套用到所有 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知

- 接收多個區域的 CloudTrail 日誌檔案及接收多個帳戶的 CloudTrail 日誌檔案

所有亞馬遜重新認知動作都由CloudTrail並記錄在亞馬遜重新認知 API 參考。例如，對 `CreateCollection`、`CreateStreamProcessor` 和 `DetectCustomLabels` 動作發出的呼叫會在 CloudTrail 日誌檔案中產生項目。

每一筆事件或日誌項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否透過根或 AWS Identity and Access Management (IAM) 使用者憑證來提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail 使用者身分元素。

了解亞馬遜重新認知日誌檔項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一個或多個日誌項目。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔案並非依公有 API 呼叫追蹤記錄的堆疊排序，因此不會以任何特定順序出現。

下面的例子顯示了CloudTrail具有下列 API 動作的記錄項目：`StartLabelDetection`和`DetectLabels`。

```
{
    "Records": [

        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/JorgeSouza",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
```

```
                        "accountId": "111122223333",
                        "userName": "Admin"
                    },
                    "webIdFederationData": {},
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2020-06-30T20:10:09Z"
                    }
                }
            },
            "eventTime": "2020-06-30T20:42:14Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "StartLabelDetection",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/3",
            "requestParameters": {
                "video": {
                    "s3Object": {
                        "bucket": "my-bucket",
                        "name": "my-video.mp4"
                    }
                }
            },
            "responseElements": {
                "jobId":
"653de5a7ee03bd5083edde98ea8fce5794fcea66d077bdd4cfb39d71aff8fc25"
            },
            "requestID": "dfcef8fc-479c-4c25-bef0-d83a7f9a7240",
            "eventID": "b602e460-c134-4ecb-ae78-6d383720f29d",
            "readOnly": false,
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/JorgeSouza",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
```

```
                        "type": "Role",
                        "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    },
                    "webIdFederationData": {},
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2020-06-30T21:19:18Z"
                    }
                }
            },
            "eventTime": "2020-06-30T21:21:47Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "DetectLabels",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/3",
            "requestParameters": {
                "image": {
                    "s3Object": {
                        "bucket": "my-bucket",
                        "name": "my-image.jpg"
                    }
                }
            },
            "responseElements": null,
            "requestID": "5a683fb2-aec0-4af4-a7df-219018be2155",
            "eventID": "b356b0fd-ea01-436f-a9df-e1186b275bfa",
            "readOnly": true,
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

Amazon Rekognition 中的準則和配額

以下章節提供在使用 Amazon Rekognition 時的指導方針和配額。有兩種配額。設定配額 (例如最大映像大小) 無法變更。您可以依照 預設配額 章節中所述的程序，變更 AWS 服務配額 頁面上列出的預設配額。

主題

受支援區域

如需提供 Amazon Rekognition 的 AWS 區域清單，請參閱亞馬 Amazon Web Services 一般參考中的 AWS 區域和端點。

設定配額

以下是 Amazon Rekognition 中無法變更的相關限制清單。如需有關每秒交易 (TPS) 限制的資訊，請參閱 預設配額。

如需 Amazon Rekognition 自訂標籤限制，請參閱 Amazon Rekognition 自訂標籤中的指導方針和配額。

Amazon Rekognition Image

- 作為 Amazon S3 物件儲存的最大映像大小為 15 MB。
- 對於高度與寬度而言 `DetectModerationLabels` 的最大映像大小為 10K 像素。
- 對於高度與寬度而言 `DetectLabels` 最大映像大小為 10K 像素。
- 在 1920X1080 像素的映像中，臉部必須大於或等於 40x40 像素才能偵測得到。大小高於 1920X1080 像素的映像需要按比例放大最小臉部大小。
- 對於高度與寬度的最小映像為 80 像素。對於高度與寬度而言 `DetectProtectiveEquipment` 的最小映像大小為 64 像素。
- 對於高度與寬度而言 `DetectProtectiveEquipment` 的最小映像大小為 4096 像素。

- 在 800x1300 像素的映像中，臉部必須不小於 100x100 像素才能被 `DetectProtectiveEquipment` 偵測到。高於 1920X1080 像素的映像需要按比例放大最小臉部大小。
- 以原始位元組傳入 API 做為參數的最大映像大小為 5 MB。`DetectProtectiveEquipment` API 的限制值為 4 MB。
- Amazon Rekognition 支援 PNG 與 JPEG 映像格式。也就是說，您提供做為各種 API 操作 (例如 `DetectLabels` 與 `IndexFaces`) 輸入的映像必須使用其中一種支援的格式。
- 一個臉部集合中最多可存放 2,000 萬張臉。
- 一個部集合中最多可存放 2,000 萬張臉。
- 搜尋 API 最多傳回 4096 張相符的臉部。
- 搜尋 API 最多傳回 4096 張相符的臉部。
- `DetectText` 最多可在一個映像中偵測到 100 個字。
- `DetectProtectiveEquipment` 最多可以偵測到 15 人的個人防護裝備。

如需映像與臉部比較的最佳實務資訊，請參閱感應器、輸入映像和影片的最佳實務。

Amazon Rekognition 圖像批量分析

- Amazon Rekognition 影像批量分析可分析影像批次，最多可分析 10 萬張大小的影像。
- Amazon Rekognition 圖像批量分析支援最大 50MB 的輸入資訊清單。

Amazon Rekognition Video 存儲視頻

- Amazon Rekognition Video 可以分析存儲的視頻大小高達 10 GB。
- Amazon Rekognition Video 可分析長度最多 6 小時的已存放影片。
- Amazon Rekognition Video 支援每個帳戶最多 20 個並行工作。
- 已存放影片必須使用 H.264 轉碼器來編碼。支援的檔案格式為 MPEG-4 與 MOV。
- 任何可分析音訊資料的 Amazon Rekognition Video API 只支援 AAC 音訊轉碼器。
- 適用於分頁符記的轉碼器 (TTL) 期間為 24 小時。分頁符記位於 Get 操作 (例如 `GetLabeldetection`) 傳回的 `NextToken` 欄位中。

Amazon Rekognition Video 頻

- Kinesis Video 輸入串流最多可與 1 個 Amazon Rekognition Video 串流處理器連接。
- Kinesis Data 輸出串流最多可與 1 個 Amazon Rekognition Video 串流處理器連接。
- 與 Amazon Rekognition Video 串流處理器相關的 Kinesis Video 輸入串流與 Kinesis Data 輸出串流無法供多個處理器共用。
- 任何可分析音訊資料的 Amazon Rekognition Video API 只支援 ACC 音訊轉碼器。

預設配額

可以在 AWS 服務配額中找到預設配額清單。這些預設限制可以變更。如需申請提高限制，請建立案例。若要查看您目前的配額限制 (套用的配額值)，請參閱 Amazon Rekognition 服務配額。若要檢視 Amazon Rekognition Image APIs 的 TPS 使用歷史記錄，請參閱 Amazon Rekognition 服務配額頁面，然後選擇特定的 API 操作以查看該作業的歷史記錄。

主題

計算 TPS 配額變更

您要求的新限制數目是多少？ 每秒交易次數 (TPS) 在預期工作負載的尖峰時最為相關。應了解工作負載尖峰期和回應時間的最大並行 API 呼叫次數 (5 至 15 秒)。請注意，5 秒應該是最短的時間。以下是兩個範例：

- 範例 1：最繁忙時段開始時，我預期的最大並行臉部驗證 (CompareFaces API) 使用者數目為 1000 人。這些回應將分散在 10 秒內。因此，我相關地區的 CompareFaces API 所需的 TPS 是 100 (1000/10)。
- 示例 2：在我最繁忙的時間開始時，預期的最大並發對象檢測（DetectLabels API）調用為 250。這些回應將分散在 5 秒內。因此，我相關地區的 DetectLabels API 所需的 TPS 是 50（250/5）。

TPS 配額的最佳實務

建議的每秒交易數 (TPS) 最佳作法包括平穩流量峰值、設定重試，以及設定指數退避和抖動。

1. 流量削峰。流量達到峰值時，會影響輸送量。若要取得每秒交易次數 (TPS) 的最大分配輸送量，請使用佇列無伺服器架構或其他機制將「流量削峰」流量，使流量更加一致。如需使用 Rekognition 進行無伺服器大規模映像和視訊處理的程式碼範例和參考，請參閱使用 Amazon Rekognition 進行大規模映像和視訊處理。
2. 設定重試。請遵循 the section called “錯誤處理” 的指導方針來設定這些錯誤的允許重試次數。
3. 設定指數退避和抖動 在設定重試時設定指數退避和抖動可讓您改善可達成的輸送量。請參閱中的錯誤重試和指數輪詢。 AWS

建立案例以變更 TPS 配額

若要建立案例，請前往建立案例，並回答下列問題：

- 您是否實施了 the section called “TPS 配額的最佳實務” 用于流量削峰並配置重試、指數退避和抖動？
- 您是否已計算所需的 TPS 配額變更？ 如果沒有，請參閱 the section called “計算 TPS 配額變更”。
- 為了更準確地預測未來的需求，您是否檢查過 TPS 使用歷史記錄？ 若要檢視您的 TPS 使用歷史記錄，請參閱 Amazon Rekognition 服務配額頁面。
- 您的使用案例是什麼？
- 您打算使用哪些 API？
- 您打算用於這些 API 的哪個區域？
- 您是否能夠將負載分散到多個區域中？
- 您每天處理多少映像？
- 您預計此流量會持續多久 (是一次性達到峰值還是持續)？
- 您如何通過預設限制來阻止？ 複查下列例外表格，以確認您遇到的情境。

錯誤代碼	異常情形	訊息	那代表什麼意思？	可否重試？
HTTP 狀態碼 400	ProvisionedThroughputExceededException	超過佈建率。	表示限流。您可以重試或評估提高限制的要求。	是
HTTP 狀態碼 400	ThrottlingException	減慢；請求率突然增加。	您可能正在發送峰值流量並遇到	是

錯誤代碼	異常情形	訊息	那代表什麼意思？	可否重試？
			限流。您應該規劃流量並使其更加平穩和一致。然後設定其他重試次數。請參閱最佳實務。	
HTTP 狀態碼 5xx	ThrottlingException（HTTP 500）	服務無法使用	指出後端正在向上擴展以支援動作。您應重試請求。	是

如需錯誤代碼的詳細瞭解，請參閱 the section called “錯誤處理”。

> **Note**
>
> 這些限制取決於您所在的地區。在您提出請求的區域中，提出更改限制的案例會影響您請求的 API 操作，。其他 API 操作和地區不受影響。

Amazon Rekognition 的文件歷史記錄

下表說明每個 Amazon Rekognition 開發人員指南版本的重要變更。如需有關此文件更新的通知，您可以訂閱 RSS 訂閱源。

- 最新文件更新時間：2023 年 6 月 15 日

變更	描述	日期
Amazon Rekognition 現在支援新的協調標籤，並改善影像內容審核的準確性	Amazon Rekognition 的內容審核功能已經過增強，可提高準確性、偵測新標籤，以及識別動畫和/或插圖內容的能力。	2024年2月1日
Amazon Rekognition 現在支援批量映像分析	Amazon Rekognition 現在支援透過將資訊清單檔案與作業搭配使用，以非同步方式處理大量映像集合。StartMediaAnalysisJob	2023 年 10 月 23 日
Amazon Rekognition 現在支援使用轉接器進行自訂內容管制	Amazon Rekognition 現在透過使用可擴充現有 Rekognition 深度學習模型功能的配接器來支援增強 DetectModerationLabels API 的準確性。	2023 年 10 月 12 日
Rekognition 現在支援具有集合的使用者向量	Rekognition 臉部集合現在支援建立使用者向量。使用者向量彙總同一使用者的多個臉部向量，藉由更可靠的使用者描述來提高準確性。	2023 年 6 月 12 日
與管理使用者有關的動作已新增至下列受管理策略：AmazonRekognitionReadOnlyAccess	Amazon Rekognition 在 `AmazonRekognitionReadOnlyAccess` 受管政策中新增了下列動作：`ListUsers`	2023 年 6 月 12 日

	、SearchUsers 、和 SearchUsersByImage	
Amazon Rekognition Image 現在可以推斷凝視方向	Amazon Rekognition Image 的臉部偵測操作已經進行了改進，現在可以推斷偵測到的臉部的凝視方向。	2023 年 5 月 31 日
Rekognition 內容管制 API 已改進	Rekognition 改善了映像和視訊管制的內容管制模型。這項改善顯著擴大了對露骨、暴力和暗示性內容的偵測。客戶現在可以更準確地偵測露骨和暴力內容，以改善使用者體驗、保護其品牌識別，並確保所有內容都符合其產業法規和政策。	2023 年 5 月 9 日
Amazon Rekognition Image 現在可以偵測被遮擋的臉部	Amazon Rekognition Image 現在可以偵測到臉部的遮擋情況。Amazon Rekognition 映像檔 DetectFaces 和 IndexFaces API 會傳回一個新 FaceOccluded 屬性，指出影像中的臉部是否因重疊的物件、衣服和身體部位而部分被部分擷取或無法完全看見。	2023 年 5 月 5 日
Rekognition 現在可以偵測臉部活體	Amazon Rekognition Video 現在可用於偵測視訊中的活體，並確認攝影機前的使用者實際存在。臉部活體檢測器還可以檢測呈現於攝影機的欺騙攻擊或試圖繞過攝影機的操作。	2023 年 4 月 11 日

更新到 Amazon Rekognition Video。	Amazon Rekognition Video 現在可以偵測更多標籤，並傳回有關映像和標籤屬性的詳細資訊。 GetLabelDetection API 現在會傳回別名和類別的相關資訊。返回的標籤信息可以使用包含性和獨佔過濾器選項進行過濾。結果可以依時間戳記或影片區段彙總。	2022 年 12 月 7 日
更新到 Amazon Rekognition Image。	Amazon Rekognition Image 現在可以偵測更多標籤，現在可以傳回有關映像和標籤屬性的詳細資訊。 DetectLabels API 現在會傳回別名、類別和影像屬性 (例如主色) 的相關資訊。返回的標籤信息可以使用包含性和獨佔過濾器選項進行過濾。	2022 年 11 月 11 日
已將「自訂標籤模型複製」的動作 ProjectPolicy 和「自訂標籤模型複製」新增至下列受管 AmazonRekognitionReadOnlyAccess	Amazon Rekognition 將下列動作新增至 AmazonRekognitionReadOnlyAccess 受管政策：ListProjectPolicies	2022 年 7 月 21 日
已將「自訂標籤模型複製」的動作 ProjectPolicy 和「自訂標籤模型複製」新增至下列管理策略 AmazonRekognitionFullAccessAmazonRekognitionCustomLabelsFullAccess	Rekognition 將下列動作新增至 AmazonRekognitionCustomLabelsFullAccess 和 AmazonRekognitionFullAccess 受管政策：CopyProjectVersion 、PutProjectPolicy 、ListProjectPolicies 和 DeleteProjectPolicy	2022 年 7 月 21 日

Amazon Rekognition Video 現在可以檢測流視頻中的標籤	Amazon Rekognition Video 可以偵測串流視訊中的標籤，例如寵物和包裹。這是使用 `CreateStreamProcessor` 操作創建的流處理器上的 `ConnectedHome` 設置來完成。	2022 年 4 月 28 日
API 參考已移出 Amazon Rekognition 開發人員指南	Amazon Rekognition API 參考現在见于《Amazon Rekognition API 参考》https://docs.aws.amazon.com/rekognition/latest/APIReference/Welcome.html。	2022 年 2 月 24 日
下列受管政策的資料集管理更新：AWS 受管政策： AmazonRekognitionReadOnlyAccess、AWS 受管政策： AmazonRekognitionFullAccess 和 AWS 受管政策： AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition 在、和`AmazonRekognitionCustomLabelsFullAccess` 受管政策中新增了下列動作：`AmazonRekognitionReadOnlyAccess` `AmazonRekognitionFullOnlyAccess` 、`CreateDataset` 、、、、`ListDatasetEntries` 、、`ListDatasetLabels` 、`DescribeDataset` `UpdateDatasetEntries` `DistributeDatasetEntries` DeleteDataset	2021 年 11 月 1 日

目錄中的新節點顯示 Amazon Rekognition 範例 GitHub	程式碼範例儲存庫中的更新 AWS 程式碼範例現在會顯示在 Amazon Rekognition 開發人員指南的個別節點中，以便於存取。	2021 年 10 月 22 日
Amazon Rekognition 可以偵測影片區段中的黑框和主要程式內容	Amazon Rekognition 可以使用 `StartSegmentDetection` 和 `GetSegmentDetection` 操作，將黑色边框、色條、片頭字幕、片尾字幕、工作室標誌和主要計畫內容識別為影片中的技術提示。	2021 年 6 月 7 日
下列受管政策的資料集管理更新：	您可以使用 Amazon Rekognition `DetectText` 操作來偵測映像中最多 100 個字。	2021 年 5 月 21 日
和的標籤更AmazonRekognitionReadOnlyAccess新AmazonRekognitionFullAccess	Rekognition 將新的標記動作新增至 `AmazonRekognitionFullAccess` 和 `AmazonRekognitionReadOnlyAccess` 政策。	2021 年 4 月 15 日
Amazon Rekognition 現在支援標記	您現在可以使用標籤來識別、組織、搜尋和篩選 Amazon Rekognition 集合、串流處理器和自訂標籤模型。	2021 年 3 月 25 日
Amazon Rekognition 在可以偵測個人防護設備	Amazon Rekognition 現在可以偵測映像中人物的手套、面罩和頭套。	2020 年 10 月 15 日
Amazon Rekognition 有新的內容管制類別	Amazon Rekognition 內容管制類別現在包括 6 個新類別：毒品、煙草、酒精、賭博、粗魯手勢和仇恨符號。	2023 年 10 月 12 日

Amazon Rekognition Video 本機顯示 Kinesis Video Streams 結果的新教程	您可以在 Kinesis Video Streams 中以本機視訊摘要的形式顯示來自串流視訊的 Amazon Rekognition Video 輸出。	2020 年 7 月 20 日
使用 Gstreamer 的 Amazon Rekognition 新教程	使用 Gstreamer，您可以透過 Kinesis Video Streams，將裝置攝影機來源的即時串流影片導入 Amazon Rekognition Video。	2020 年 7 月 17 日
Amazon Rekognition 目前支援已存放影片的分鏡	使用 Amazon Rekognition Video 分鏡 API，您可以偵測已存放影片中的黑色边框、色條、片尾字幕以及鏡頭。	2020 年 6 月 22 日
Amazon Rekognition 現在支援 Amazon VPC 端點政策	藉由指定政策，您可以限制存取 Amazon Rekognition Amazon VPC 端點。	2020 年 3 月 3 日
Amazon Rekognition 現在支援偵測儲存影片中的文字	您可以使用 Amazon Rekognition Video API 來非同步偵測已儲存視訊中的文字。	2020 年 2 月 17 日
Amazon Rekognition 現在支援 Augmented AI (預覽) 和 Amazon Rekognition 自訂標籤	透過 Amazon Rekognition 自訂標籤，您可以藉由建立自己的機器學習模型，來偵測映像中的特殊物件、場景和概念。DetectModerationLabels 現在支持 Amazon Augmented AI（預覽）。	2019 年 12 月 3 日
Amazon Rekognition 現在支援 AWS PrivateLink	使用 AWS， PrivateLink 您可以在 VPC 和 Amazon Rekognition 之間建立私有連接。	2019 年 9 月 12 日

Amazon Rekognition 臉部過濾	Amazon Rekognition 為 API 操作新增了增強的臉部篩選支援，並為和 IndexFaces API 操作引入了臉部篩選功能。CompareFaces SearchFacesByImage	2019 年 9 月 12 日
Amazon Rekognition Video 示例已更新	Amazon Rekognition Video 範例程式碼已更新，以建立和設定 Amazon SNS 主題和 Amazon SQS 佇列。	2019 年 9 月 5 日
Ruby 和 Node.js 範例已新增	針對同步標籤和臉部偵測新增的 Amazon Rekognition Image Ruby 和 Node.js 範例。	2019 年 8 月 19 日
不安全內容偵測已更新	Amazon Rekognition 不安全內容偵測現在可以偵測暴力內容。	2019 年 8 月 9 日
GetContentModeration 操作已更新	GetContentModeration 現在會傳回用來偵測不安全內容的協調偵測模型版本。	2019 年 2 月 13 日
GetLabelDetection 和 DetectModerationLabels 操作已更新	GetLabelDetection 現在會傳回常用物件的邊界方塊資訊，以及偵測到之標籤的階層分類法。現在會傳回用於標籤偵測的模型版本。 DetectModerationLabels 現在會傳回用於偵測不安全內容的模型版本。	2019 年 1 月 17 日

DetectFaces 和 IndexFaces 操作更新	此版本會更新 DetectFaces 和 IndexFaces 作業。當「屬性」輸入參數設定為「全部」時，臉部位置標記會包含 5 個新地標： upperJawlineLeft、midJawlineLeft、chinBottom、midJawlineRight、 upperJawlineRight。	2018 年 11 月 19 日
DetectLabels 操作已更新	現在，對於特定物件會傳回週框方塊。階層式分類現在可用於標籤。您現在可以取得用於偵測的偵測模型版本。	2018 年 11 月 1 日
IndexFaces 操作已更新	使用時， IndexFaces 您現在可以使用 QualityFilter 輸入參數過濾掉低質量檢測到的臉孔。您也可以使用 MaxFaces 輸入參數，根據臉部偵測的品質以及偵測到的臉部大小來減少傳回的臉部數目。	2018 年 9 月 18 日
DescribeCollection 已加入作業	您現在可以呼叫 DescribeCollection 作業，取得現有集合的相關資訊。	2018 年 8 月 22 日
新的 Python 範例	Python 範例已新增到 Amazon Rekognition Video 內容以及一些內容重組。	2018 年 6 月 26 日
更新內容版面配置	Amazon Rekognition Image 內容已重新整理過，並有新增的 Python 與 C# 範例。	2018 年 5 月 29 日

Amazon Rekognition 支援 AWS CloudTrail	Amazon Rekognition 已與 AWS CloudTrail 整合，這項服務可提供由使用者、角色或 Amazon Rekognition 中 AWS 服務所採取之動作的記錄。如需詳細資訊，請參閱使用 AWS 記錄 Amazon Rekognition API 呼叫。 CloudTrail	2018 年 4 月 6 日
分析已存放與串流的影片。新目錄	如需分析已儲存影片的相關資訊，請參閱使用儲存的影片。如需分析串流影片的相關資訊，請參閱使用串流影片。Amazon Rekognition 文件的目錄已經過重新編排，以納入映像與影片操作的說明。	2017 年 11 月 29 日
映像與臉部偵測模型中的文字	Amazon Rekognition 現在可以偵測映像中的文字。如需更多詳細資訊，請參閱偵測文字。Amazon Rekognition 推出臉部偵測深度學習模型的版本控制。如需詳細資訊，請參閱模型版本控制。	2017 年 11 月 21 日
名人辨識	Amazon Rekognition 現在可以分析名人的映像。如需詳細資訊，請參閱辨識名人。	2017 年 6 月 8 日
映像管制	Amazon Rekognition 現在可以判斷映像是否含有露骨或暗示性的成人內容。如需詳細資訊，請參閱刪除不安全的內容。	2017 年 4 月 19 日

偵測到臉部的人物的年齡範圍。彙總的 Rekognition 指標窗格	Amazon Rekognition 現在會傳回 Rekognition API 偵測到之臉部所估計的年齡範圍 (以歲為單位)。如需詳細資訊，請參閱 AgeRange。Rekognition 主控台現在有一個指標窗格，顯示指定時間段內 Rekognition 彙總 Amazon CloudWatch 指標的活動圖表。如需詳細資訊，請參閱練習 4：請參閱彙總指標 (主控台)。	2017 年 2 月 9 日
新的服務與指南	這是映像分析服務 Amazon Rekognition 及 Amazon Rekognition 開發人員指南的初始版本。	2016 年 11 月 30 日

AWS 詞彙表

如需最新的 AWS 術語，請參閱《AWS 詞彙表 參考》中的 AWS 詞彙表。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。