aws

개발자 가이드

Amazon Rekognition

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Rekognition: 개발자 가이드

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께 사용되어서는 안되며, 고객에게 혼동을 일으키거나 Amazon 브랜드 이미지를 떨어뜨리고 폄하하는 방식으로 이용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

Table of Contents

Amazon Rekognition이란 무엇인가요? 1
주요 기능 1
사용 사례 2
이점 3
Amazon Rekognition과 HIPAA 자격 4
Amazon Rekognition을 처음 사용하시나요? 4
Rekognition 작동 방식 5
Rekognition의 분석 유형 이해 6
레이블 8
사용자 지정 레이블 9
Face Liveness 감지 9
얼굴 감지 및 분석 10
얼굴 검색 10
인물 경로 10
개인용 보호 장비 11
유명 인사 11
텍스트 감지 11
부적절하거나 불쾌감을 주는 콘텐츠 11
사용자 지정 11
대량 분석 12
Rekognition의 이미지 및 비디오 작업 이해 12
Amazon Rekognition Image 작업 12
Amazon Rekognition Video 작업 13
비스토리지 및 스토리지 기반 작업 13
AWS SDK 또는 HTTP를 사용하여 Amazon Rekognition API 작업 직접 호출 14
비스토리지 및 스토리지 API 작업 14
비스토리지 작업 14
스토리지 기반 API 작업 16
모델 버전 관리 이해 18
시작하기 20
1단계: AWS 계정 설정 및 사용자 생성 20
AWS 계정 및 사용자 생성 21
2단계: AWS CLI 및 AWS SDKs 설정 23
프로그래밍 방식 액세스 권한 부여 25

AWS SDKs 작업 29
3단계: AWS CLI 및 AWS SDK API 사용 시작하기 30
AWS CLI 예제 형식 지정 30
다음 단계 31
4단계: 콘솔을 사용하여 시작 31
콘솔 권한 설정 32
연습 1: 객체 및 장면 감지(콘솔) 35
연습 2: 얼굴 분석(콘솔) 42
연습 3: 얼굴 비교(콘솔) 45
연습 4: 집계 지표 보기(콘솔) 48
이미지 및 비디오 작업 50
이미지 작업 50
이미지 사양 51
Amazon S3 버킷의 이미지 분석 52
로컬 파일 시스템 사용 69
경계 상자 표시 84
이미지 방향 및 경계 상자 좌표 가져오기 96
저장된 비디오 분석 작업 107
분석 유형 108
Amazon Rekognition Video API 개요 108
Amazon Rekognition Video 작업 직접 호출 111
Amazon Rekognition Video 구성 117
저장된 비디오 분석(SDK) 120
비디오 분석(AWS CLI) 149
참조: 비디오 분석 결과 알림 153
Amazon Rekognition Video 문제 해결 154
스트리밍 비디오 이벤트 작업 156
Amazon Rekognition Video 스트림 프로세서 작업 개요 156
새 스트림 프로세서에 태그 추가 157
기존 스트림 프로세서에 태그 추가 158
스트림 프로세서의 태그 나열 158
스트림 프로세서에서 태그 삭제 159
오류 처리 159
오류 구성 요소 160
오류 메시지 및 코드 160
애플리케이션에서의 오류 처리 165

FedRAMP와 함께 Amazon Rekognition 사용 166
센서, 입력 이미지 및 비디오 모범 사례 170
Amazon Rekognition Image 작업 지연 시간 170
얼굴 비교 입력 이미지에 대한 권장 사항 171
얼굴 작업을 위한 입력 이미지에 대한 일반 권장 사항 171
컬렉션에서 얼굴을 검색할 때의 권장 사항 172
카메라 설정 권장 사항(이미지 및 비디오) 172
카메라 설정 권장 사항(저장 및 스트리밍 비디오) 174
카메라 설정 권장 사항(스트리밍 비디오) 175
Face Liveness 사용에 대한 권장 사항 175
객체 및 개념 감지 177
레이블 응답 객체 178
경계 상자 178
신뢰도 점수 179
상위 개념 179
Categories 180
별칭 180
이미지 속성 180
모델 버전 182
포함 및 제외 필터 182
결과 정렬 및 집계 183
이미지에서 레이블 감지 183
DetectLabels 작업 요청 194
DetectLabels 응답 195
DetectLabels 응답의 변환 199
비디오에서 레이블 감지 203
StartLabelDetection 요청 203
GetLabelDetection 작업 응답 205
GetLabelDetection 응답의 변환 211
스트리밍 비디오 이벤트의 레이블 감지 219
Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정 220
스트리밍 비디오 이벤트에 대한 레이블 감지 작업 직접 호출 224
사용자 지정 레이블 감지 230
얼굴 감지 및 분석 231
얼굴 감지 및 얼굴 비교 개요 232
얼굴 속성에 대한 가이드라인 233

이미지에서 얼굴 감지 234
DetectFaces 작업 요청 246
DetectFaces 작업 응답 246
이미지에 있는 얼굴 비교 253
CompareFaces 작업 요청 266
CompareFaces 작업 응답 266
저장된 비디오에서 얼굴 감지 269
GetFaceDetection 작업 응답 278
컬렉션에서 얼굴 검색 284
얼굴 컬렉션, 얼굴 및 사용자 관리 287
컬렉션 관리 287
컬렉션에서 얼굴 관리 288
컬렉션의 사용자 관리 288
얼굴 연결 및 매칭을 위한 유사성 임곗값 사용 289
얼굴 연결을 위한 유사성 임계값 사용 289
유사성 임계값을 사용하여 얼굴 일치 289
일반적인 시나리오에서 얼굴 인덱싱 지침 290
중요 또는 공공 안전 애플리케이션 290
사진 공유 및 소셜 미디어 애플리케이션 290
일반 사용 사례 290
컬렉션 내의 얼굴 및 사용자 검색 290
공공 안전 관련 사용 사례 291
공공 안전을 위한 Amazon Rekognition 사용 292
컬렉션 생성 293
CreateCollection 작업 요청 299
CreateCollection 작업 응답 300
컬렉션 태그 지정 300
새 컬렉션에 태그 추가 300
기존 컬렉션에 태그 추가 301
컬렉션의 태그 나열 303
컬렉션에서 태그 삭제 304
컬렉션 나열 305
ListCollections 작업 요청 311
ListCollections 작업 응답 312
컬렉션 설명 312
DescribeCollection 작업 요청 319

DescribeCollection 작업 응답 319
컬렉션 삭제 320
DeleteCollection 작업 요청 326
DeleteCollection 작업 응답 326
컬렉션에 얼굴 추가 326
얼굴 필터링 328
IndexFaces 작업 요청 337
IndexFaces 작업 응답 337
컬렉션의 얼굴 및 연결된 사용자 나열 346
ListFaces 작업 요청 352
ListFaces 작업 응답 352
컬렉션에서 얼굴 삭제 354
DeleteFaces 작업 요청 359
DeleteFaces 작업 응답 359
사용자 생성 360
사용자 삭제 364
얼굴을 사용자에 연결 366
AssociateFaces 작업 응답 369
사용자에게서 얼굴 연결 해제 370
DisassociateFaces 작업 응답 373
컬렉션에서 사용자 나열 374
ListUsers 작업 응답 377
얼굴 검색(얼굴 ID) 378
SearchFaces 작업 요청 384
SearchFaces 작업 응답 384
얼굴(이미지) 검색 385
SearchFacesByImage 작업 요청 393
SearchFacesByImage 작업 응답 393
사용자 검색(얼굴 ID 또는 사용자 ID) 395
SearchUsers 작업 요청 398
SearchUsers 작업 응답 399
사용자 검색(이미지) 400
SearchUsersByImage 작업 요청 404
SearchUsersByImage 작업 응답 404
저장된 비디오에서 얼굴 검색 406
GetFaceSearch 작업 응답 415

스트리밍 비디오에서 컬렉션의 얼굴 검색 420
Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정 420
스트리밍 비디오에서 얼굴 검색 423
GStreamer 플러그인을 사용한 스트리밍 446
스트리밍 비디오 문제 해결 448
인물 경로 추적 455
GetPersonTracking 작업 응답 464
개인 보호 장비 감지 469
PPE의 유형 470
얼굴 덮개 470
손 덮개 470
머리 덮개 470
PPE 탐지 신뢰도 470
이미지에서 감지된 PPE 요약 471
자습서: PPE를 사용하여 이미지를 감지하는 AWS Lambda 함수 생성 471
PPE 탐지 API 이해 472
이미지 제공 472
DetectProtectiveEquipment 응답 이해 473
이미지에서 PPE 감지 479
예제: 경계 상자와 얼굴 덮개 491
유명 인사 인식 506
얼굴 검색과 유명 인사 인식 비교 506
이미지 속 유명 인사 인식 507
RecognizeCelebrities 호출 507
RecognizeCelebrities 작업 요청 517
RecognizeCelebrities 작업 응답 517
저장된 동영상 속 유명 인사 인식 520
GetCelebrityRecognition 작업 응답 535
유명 인사 정보 얻기 538
GetCelebrityInfo 호출 538
GetCelebrityInfo 작업 요청 543
GetCelebrityInfo 작업 응답 543
콘텐츠 조절 544
이미지 및 비디오 조절 API 사용 545
레이블 카테고리 546
콘텐츠 유형 556

신뢰도 556
버전 관리 557
정렬 및 집계 557
사용자 지정 조절 어댑터 상태 557
콘텐츠 조절 버전 7 테스트 및 API 응답 변환 558
AWS 콘텐츠 조절 버전 7용 SDK 및 사용 설명서 558
버전 6.1에서 7로 레이블 매핑 559
부적절한 이미지 감지 564
이미지에서 부적절한 콘텐츠 감지 564
DetectModerationLabels 작업 요청 571
DetectModerationLabels 작업 응답 571
부적절한 저장된 동영상 탐지 573
GetContentModeration 작업 응답 581
사용자 지정 조절을 통한 정확도 향상 584
어댑터 생성 및 사용 584
데이터 세트 준비 588
AWS CLI 및 SDKs를 사용하여 어댑터 관리 590
사용자 지정 조절 어댑터 자습서 596
어댑터 평가 및 개선 614
매니페스트 파일 형식 616
어댑터 훈련 모범 사례 621
자동 업데이트 권한 설정 621
AWS Rekognition에 대한 Health Dashboard 인증 624
Amazon A2I를 사용한 부적절한 콘텐츠 검토 625
텍스트 감지 631
이미지에서 텍스트 감지 633
DetectText 작업 요청 641
DetectText 작업 응답 642
저장된 비디오에서 텍스트 감지 647
필터 657
GetTextDetection 응답 658
비디오 세그먼트 감지 664
기술적 큐 665
블랙 프레임 665
Credits 665
색상 막대 665

슬레이트 ... 666
스튜디오 로고 ... 666
내용 ... 666
샷 감지 ... 667
Amazon Rekognition Video 세그먼트 탐지 API 정보 ... 668
Amazon Rekognition Segment API 사용 ... 668
세그먼트 분석 시작 ... 668
세그먼트 분석 결과 가져오기 ... 669
예제: 저장된 비디오에서 세그먼트 감지 ... 674
얼굴 생체 확인 감지 ... 687
사용자 측 얼굴 생체 확인 요구 사항 ... 689
아키텍처 및 시퀀스 다이어그램 ... 690
사전 조건 ... 692
1단계: AWS 계정 설정 ... 692
2단계: Face AWS SDKs 설정 ... 692
3단계: AWS Amplify 리소스 설정 ... 693
Face Liveness 감지 모범 사례 ... 693
Amazon Rekognition Face Liveness API 프로그래밍 ... 693
1단계: CreateFaceLivenessSession ... 694
2단계: StartFaceLivenessSession ... 695
3단계: GetFaceLivenessSessionResults ... 695
4단계: 결과에 응답 ... 696
Face Liveness API 직접 호출 ... 696
애플리케이션 구성 및 사용자 지정 ... 703
애플리케이션 구성 ... 703
애플리케이션 사용자 지정 ... 703
Face Liveness 공동 책임 모델 ... 703
Face Liveness 업데이트 가이드라인 ... 708
버전 관리 및 기간 범위 ... 708
버전 출시 및 호환성 매트릭스 ... 709
새 릴리스에 대한 알림 ... 709
Face Liveness FAQ ... 709
대량 분석 ... 713
대량 이미지 처리 ... 713
대량 분석 작업(CLI)을 만들려면 ... 713
대량 분석 출력 매니페스트 ... 715

콘텐츠 유형 716
예측 확인 및 어댑터 훈련 716
자습서 717
Amazon RDS 및 DynamoDB로 Amazon Rekognition 데이터 저장 717
사전 조건 718
Amazon S3 버킷에 있는 이미지에 대한 레이블 가져오기 718
Amazon DynamoDB 테이블 생성 720
DynamoDB로 데이터 업로드 721
Amazon RDS에서 MySQL 데이터베이스 생성 723
Amazon RDS MySQL 테이블에 데이터 업로드 724
Amazon Rekognition과 Lambda를 사용하여 Amazon S3 버킷의 자산에 태그 지정 726
사전 조건 728
IAM Lambda 역할 구성 728
프로젝트 생성 729
코드 쓰기 731
프로젝트 패키지화 742
Lambda 함수 배포 743
Lambda 메서드 테스트 743
AWS 비디오 분석기 애플리케이션 생성 744
사전 조건 745
절차 746
Amazon Rekognition Lambda 함수 생성 746
사전 조건 748
SNS 주제 생성 748
Lambda 함수 생성 748
Lambda 함수 구성 749
IAM Lambda 역할 구성 750
AWS Toolkit for Eclipse Lambda 프로젝트 생성 750
Lambda 함수 테스트 754
신원 확인을 위한 Amazon Rekognition 사용 756
사전 조건 756
모음 만들기 757
신규 사용자 등록 758
기존 사용자 로그인 766
Amazon Rekognition(REK)을 사용하여 마케팅 애플리케이션의 레이블 감지 768
사전 조건 769

DynamoDB 테이블 생성 및 채우기 770
로깅 및 알림을 사용하여 Lambda 함수 생성 772
권한 및 Lambda 함수 설정 774
메타데이터를 사용하여 Amazon S3에 이미지 업로드 776
클라이언트에서 CloudWatch 로그 폴링 777
Lambda와 Python을 사용하여 이미지에서 레이블 감지 779
Lambda 함수 생성(콘솔) 779
(선택 사항) 계층 생성(콘솔) 781
Python 코드 추가(콘솔) 782
Python 코드를 추가하려면(콘솔) 784
코드 예제 788
기본 사항 789
Amazon Rekognition 시작 790
작업 793
시나리오 961
컬렉션 구축 및 컬렉션에서 얼굴 검색 962
사진을 관리하기 위한 서버리스 애플리케이션 만들기 974
이미지에서 PPE 감지 978
이미지에서 요소 감지 및 표시 979
이미지에서 얼굴 감지 995
동영상 내 정보 감지 995
이미지에서 객체 감지 1035
동영상에서 사람과 객체 감지 1038
EXIF 및 기타 이미지 정보 저장 1039
API 참조 1041
보안 1042
ID 및 액세스 관리 1042
대상 1043
ID를 통한 인증 1043
정책을 사용하여 액세스 관리 1044
Amazon Rekognition에서 IAM을 사용하는 방법 1045
AWS 관리형 정책 1051
정책 모범 사례 1060
콘솔 사용 1060
자격 증명 기반 정책 예제 사용의 예시 1061
리소스 기반 정책의 예 1064

문제 해결 1065
데이터 보호 1067
데이터 암호화 1067
인터네트워크 트래픽 개인 정보 1070
Amazon Rekognition에서 Amazon VPC 엔드포인트 사용 1070
Amazon Rekognition용 Amazon VPC 엔드포인트 생성 1071
Amazon Rekognition에 대한 VPC 엔드포인트 정책 생성 1071
규정 준수 확인 1073
복원성 1073
구성 및 취약성 분석 1074
교차 서비스 혼동된 대리인 방지 1074
인프라 보안 1075
모니터링 1076
Amazon CloudWatch를 사용한 Rekognition 모니터링 1076
Rekognition에 CloudWatch 지표 사용 1077
Rekognition 지표 액세스 1078
경보 만들기 1079
Rekognition의 CloudWatch 지표 1080
를 사용하여 Amazon Rekognition API 호출 로깅 AWS CloudTrail 1085
CloudTrail의 Amazon Rekognition 정보 1085
Amazon Rekognition 로그 파일 항목 이해 1086
지침과 할당량 1089
지원되는 리전 1089
고정된 할당량 1089
Amazon Rekognition Image 1089
Amazon Rekognition Image 대량 분석 1089
Amazon Rekognition Video 저장 동영상 1090
Amazon Rekognition Video 스트리밍 비디오 1091
기본 할당량 1091
TPS 할당량 변경 계산 1091
TPS 할당량 모범 사례 1092
TPS 할당량 변경을 위한 사례 생성 1092
문서 이력 1095
AWS 용어집 1106
..... mcvii

Amazon Rekognition이란 무엇인가요?

Amazon Rekognition은 애플리케이션에 고급 컴퓨터 비전 기능을 쉽게 추가할 수 있는 클라우드 기반 이미지 및 비디오 분석 서비스입니다. 이 서비스는 검증된 딥 러닝 기술로 구동되며 기계 학습 전문 지식이 필요하지 않습니다. Amazon Rekognition에는 Amazon S3에 저장된 이미지 또는 비디오 파일을 빠르게 분석할 수 있는 간단하고 사용하기 쉬운 API가 포함되어 있습니다.

Rekognition의 API를 사용하여 객체, 텍스트, 안전하지 않은 콘텐츠를 감지하고, 이미지/비디오를 분석하며, 얼굴을 비교하는 기능을 애플리케이션에 추가할 수 있습니다. Amazon Rekognition의 얼굴 인식 API를 이용하여 사용자 확인, 카탈로그 작성, 인원 계산 및 공공 안전을 포함한 다양한 사용 사례에서 얼굴을 탐지, 분석, 비교할 수 있습니다.

이 서비스는 매일 수십억 개의 이미지와 비디오를 분석할 수 있는 기술이자 Amazon의 컴퓨터 비전 과학자들이 개발하여 성능이 검증되었을 뿐만 아니라 확장성까지 뛰어난 딥 러닝 기술을 기반으로 하고 있습니다. Rekognition은 새 데이터에서 정기적으로 학습하며, 서비스에 새 레이블과 기능이 수시로 추가됩니다.

자세한 내용은 Amazon Rekognition FAQ를 참조하세요.

주요 기능

이미지 분석:

- 객체, 장면 및 개념 감지 - 이미지에서 객체, 장면, 개념 및 유명 인사를 감지하고 분류합니다.
- 텍스트 감지 - 이미지에서 인쇄 및 수기 텍스트를 다양한 언어로 감지하고 인식합니다.
- 안전하지 않은 콘텐츠 - 명시적, 부적절, 폭력적 콘텐츠 및 이미지를 감지하고 필터링합니다. 세분화된 안전하지 않은 콘텐츠 레이블을 감지합니다.
- 유명 인사 인식 - 정치인, 운동선수, 배우, 뮤지션 등 다양한 범주의 이미지에서 수만 명의 유명 인사를 인식합니다.
- 얼굴 분석 - 성별, 연령, 감정과 같은 얼굴 속성과 함께 얼굴을 감지, 분석 및 비교합니다. 사용 사례에는 사용자 확인, 카탈로그 작성, 인원 계산 및 공공 안전이 포함될 수 있습니다.
- 사용자 지정 레이블 - 사용자 지정 분류자를 구축하여 로고, 제품, 문자와 같은 사용 사례에 맞는 객체를 감지합니다.
- 이미지 속성 - 품질, 색상, 선명도, 대비와 같은 이미지 속성을 분석합니다.

비디오 분석:

- 객체, 장면 및 개념 감지 - 비디오에서 객체, 장면, 개념 및 유명 인사를 감지하고 분류합니다.
- 텍스트 감지 - 비디오에서 인쇄 및 수기 텍스트를 다양한 언어로 감지하고 인식합니다.
- 인물 경로 지정 - 식별된 사람이 비디오 프레임에서 이동할 때 추적합니다.
- 얼굴 분석 - 스트리밍 또는 저장된 비디오에서 얼굴을 감지, 분석 및 비교합니다.
- 유명 인사 인식 - 정치인, 운동선수, 배우, 뮤지션 등 다양한 범주의 저장된 비디오에서 수만 명의 유명 인사를 인식합니다.
- 안전하지 않은 콘텐츠 감지 - 비디오에서 명시적, 부적절 및 폭력적 콘텐츠를 감지합니다.
- 비디오 세분화 - 블랙 프레임 및 최종 크레딧과 같은 유용한 비디오 세그먼트를 자동으로 식별합니다.
- 얼굴 생체 확인 - 얼굴 인증 과정에서 실시간 사용자가 있는지 감지합니다.

사용 사례

검색 가능한 미디어 라이브러리 - Rekognition은 이미지 및 비디오에서 레이블, 객체, 개념 및 장면을 감지합니다. 이 시각적 콘텐츠 분석을 기반으로 이러한 레이블을 검색할 수 있도록 할 수 있습니다. 검색 가능한 이미지 및 비디오 라이브러리를 구축하는 데 유용합니다.

얼굴 기반 사용자 ID 확인 - 이미지의 얼굴을 참조 얼굴 이미지와 비교하여 사용자 ID를 확인합니다. 애플리케이션에서 ID 확인에 유용합니다.

얼굴 생체 확인 감지 - Rekognition Face Liveness는 개발자가 얼굴 기반 ID 확인 과정에서 사기를 방지할 수 있도록 설계된 완전 관리형 기계 학습(ML) 기능입니다. 이 기능을 사용하면 사용자가 카메라 앞에 실제로 존재하는지, 사용자의 얼굴을 도용한 악의적인 행위자는 아닌지 확인할 수 있습니다. Rekognition Face Liveness를 사용하면 인쇄된 사진, 디지털 사진 및 비디오, 또는 3D 마스크와 같이 카메라에 가해지는 스푸핑 공격을 탐지하는 데 도움을 받을 수 있습니다. 또한 비디오 캡처 하위 시스템에 직접 삽입되는 사전 녹화된 비디오 또는 딥페이크 비디오와 같이 카메라를 우회하는 스푸핑 공격을 탐지하는 데도 도움이 됩니다.

얼굴 검색 - Rekognition을 통해 얼굴 컬렉션으로 알려진 컨테이너에 저장된 것과 일치하는 얼굴의 이미지, 저장된 비디오 및 스트리밍 비디오를 검색할 수 있습니다. 얼굴 모음은 귀하께서 소유하고 관리하는 얼굴 인덱스입니다. 얼굴을 기반으로 사람을 검색하려면 얼굴을 인덱싱한 다음 얼굴을 검색해야 합니다.

안전하지 않은 콘텐츠 감지 - 이미지와 비디오에서 명시적, 부적절, 폭력적 콘텐츠를 감지하고 필터링합니다. 비즈니스 요구 사항에 따라 세분화된 필터링에 레이블을 사용합니다. Content Moderation API

는 감지된 레이블(객체 및 개념)의 계층적 목록도 신뢰도 점수와 함께 반환합니다. 이러한 객체 또는 레이블은 안전하지 않은 콘텐츠의 특정 카테고리를 가리키므로 대량의 사용자 생성 콘텐츠(UGC)의 세분화된 필터링 및 관리가 가능합니다. Content Moderation API의 결과물은 어댑터로 사용자 지정할 수 있으며, 학습 데이터로 제공하는 것 같은 이미지에서의 성능을 향상시킬 수 있습니다.

개인 보호 장비 감지 - 이미지에서 개인 보호 장비를 감지하여 다양한 산업의 안전 규정 준수를 모니터링합니다. 부적절한 장비를 감지하여 안전하지 않은 조건에 자동으로 플래그를 지정하며 이러한 조건에 대한 알림을 수신하여 규정 준수 및 훈련을 개선할 수 있습니다.

유명 인사 인식 - 정치인, 운동선수, 배우, 뮤지션 등 다양한 범주의 이미지와 비디오에서 유명 인사를 인식합니다. 이름을 제공하지 않고도 유명 인사의 모습을 식별할 수 있습니다.

텍스트 감지 - 시각적 검색 또는 메타데이터 추출을 위해 이미지의 텍스트를 감지하고 추출합니다. 이는 다양한 글꼴과 스타일에 적용됩니다. 표지판 및 배너의 텍스트를 처리하기 위한 방향을 감지합니다.

사용자 지정 레이블 - 로고 감지와 같은 비즈니스 사용 사례에 맞게 사용자 지정 객체, 개념 및 장면을 식별합니다. 사용자 지정 분류자를 훈련하여 틈새 또는 독점 객체를 처리할 수 있으므로 일반 분류자보다 키 객체의 정확도가 향상됩니다. 자세한 내용은 Amazon Rekognition Custom Labels 개발자 안내서의 Amazon Rekognition Custom Labels란 무엇인가요?를 참조하세요.

이점

앱에 강력한 이미지 및 비디오 분석 통합 - 전문 지식 없이 앱에 정확한 이미지 및 비디오 분석을 추가합니다. Amazon Rekognition API는 기계 학습 지식 없이도 딥 러닝을 통한 분석을 가능하게 합니다. 웹, 모바일 및 디바이스 앱에 컴퓨터 비전을 빠르게 구축할 수 있습니다.

딥 러닝 기반 이미지 및 비디오 분석 - 높은 정확도를 위해 딥 러닝을 사용하여 이미지 및 비디오를 분석합니다. Amazon Rekognition은 레이블, 객체, 장면, 얼굴, 유명 인사를 감지할 수 있습니다. 특정 레이블을 포함/제외하도록 결과를 필터링합니다.

확장 가능한 이미지 분석 - 수백만 개의 이미지를 분석하여 대규모 시각적 데이터 세트를 구성합니다. 증가하는 이미지 라이브러리 및 트래픽을 처리하도록 확장합니다. 용량을 계획할 필요가 없으며 사용한 만큼만 비용을 지불합니다.

속성을 기반으로 이미지 분석 및 필터링 - 품질, 색상, 시각적 콘텐츠와 같은 속성을 기준으로 이미지를 분석 및 필터링하고 이미지 선명도, 밝기, 대비를 감지합니다.

다른 AWS 서비스와의 통합 - Amazon Rekognition은 S3 및 Lambda와 즉시 통합됩니다. Lambda에서 Amazon Rekognition의 API를 직접 호출하고, 데이터를 이동하지 않고도 Amazon S3에서 이미지를 처리할 수 있습니다. Rekognition에는 AWS IAM을 사용한 확장성 및 보안이 통합되어 있습니다.

저렴한 비용 - 종량제 요금이며, 최소 금액 또는 약정이 없습니다. 프리 티어로 시작할 수 있습니다. 계층화된 요금을 통해 사용량에 따라 더 많이 절약할 수 있습니다. 사내 솔루션에 비해 비용 효과적입니다.

간단한 사용자 지정 - 어댑터를 사용하여 사용 사례에 대한 정확도를 사용자 지정합니다. 어댑터를 훈련할 샘플 이미지를 제공합니다. 지정된 도메인에 대한 객체 및 레이블 감지를 개선합니다. ML 전문 지식 없이 분석을 쉽게 조정할 수 있습니다.

자세한 내용은 Amazon Rekognition FAQ를 참조하세요.

Amazon Rekognition과 HIPAA 자격

이것은 HIPAA 적격 서비스입니다. AWS, 1996년 미국 HIPAA(Health Insurance Portability and Accountability Act) 및 AWS 서비스를 사용하여 보호되는 건강 정보(PHI)를 처리, 저장 및 전송하는 방법에 대한 자세한 내용은 HIPAA 개요를 참조하세요.

Amazon Rekognition을 처음 사용하시나요?

Amazon Rekognition을 처음 사용한다면, 다음 섹션을 순서대로 읽어보기를 권장합니다.

1. Amazon Rekognition 작동 방식 - 이 섹션에서는 종단 간 경험 생성에 사용하는 다양한 Amazon Rekognition 구성 요소를 소개합니다.
2. Amazon Rekognition 시작 - 이 섹션에서는 계정을 설정하고, 선택한 언어를 반영하여 SDK를 설치하고, Amazon Rekognition API를 테스트합니다. Amazon Rekognition에서 지원하는 프로그래밍 언어 목록은 AWS SDK에서 Rekognition 사용 섹션을 참조하세요.
3. 이미지 작업 - 이 섹션에서는 Amazon S3 버킷에 저장된 이미지와 로컬 파일 시스템에서 로드된 이미지에서 Amazon Rekognition을 사용하는 방법에 대한 정보를 다룹니다.
4. 저장된 비디오 분석 작업 - 이 섹션에서는 Amazon S3 버킷에 저장된 비디오에서 Amazon Rekognition을 사용하는 방법에 대한 정보를 다룹니다.
5. 스트리밍 비디오 이벤트 작업 - 이 섹션에서는 스트리밍 비디오에서 Amazon Rekognition을 사용하는 방법에 대한 정보를 다룹니다.

Amazon Rekognition 작동 방식

Amazon Rekognition은 시각적 분석을 위한 두 개의 API 세트를 제공합니다.

- 이미지 분석을 위한 Amazon Rekognition Image
- 비디오 분석을 위한 Amazon Rekognition Video

이미지 분석

Amazon Rekognition Image를 사용하면 애플리케이션이 다음을 수행할 수 있습니다.

- 이미지에서 객체, 장면, 개념 감지
- 유명 인사 인식
- 다양한 언어의 텍스트 감지
- 명시적, 부적절 또는 폭력적 콘텐츠 또는 이미지 감지
- 연령 및 감정과 같은 얼굴 속성과 얼굴 감지, 분석, 비교
- PPE 존재 감지

사용 사례에는 사진 앱 향상, 이미지 카탈로그 작성, 콘텐츠 조정이 포함됩니다.

비디오 분석

Amazon Rekognition Video를 사용하면 애플리케이션이 다음을 수행할 수 있습니다.

- 비디오 프레임에서 사람 및 객체 추적
- 객체 인식
- 유명 인사 인식
- 관심 인물에 대한 저장된 비디오 및 스트리밍 비디오 검색
- 얼굴에서 연령 및 감정과 같은 속성 분석
- 명시적, 부적절 또는 폭력적 콘텐츠 또는 이미지 감지
- 타임스탬프 및 세그먼트별로 분석 결과 집계 및 정렬
- 스트리밍 비디오에서 사람, 반려 동물 및 패키지 감지

사용 사례에는 비디오 분석, 비디오 카탈로그 작성, 부적절한 콘텐츠 필터링이 포함됩니다.

주요 기능

- 강력한 딥 러닝 분석
- 객체, 장면, 얼굴, 텍스트를 높은 정확도로 감지
- 앱에 통합할 수 있는 간편한 API
- 데이터에 맞게 조정된 사용자 지정 가능한 모델
- 미디어 라이브러리의 확장 가능한 분석

Amazon Rekognition을 사용하면 사용자 지정 어댑터를 훈련하여 특정 딥 러닝 모델의 정확도를 높일 수 있습니다. 예를 들어 Amazon Rekognition 사용자 지정 조절을 사용하면 사용자 지정 어댑터를 자체 이미지로 학습시켜 Amazon Rekognition의 기본 이미지 분석 모델을 조정할 수 있습니다. 자세한 내용은 사용자 지정 조절을 통한 정확도 향상을 참조하세요.

다음 섹션에서는 Amazon Rekognition에서 제공하는 분석 유형과 Amazon Rekognition Image 및 Amazon Rekognition Video 작업에 대한 개요를 다룹니다. 또한 비스토리지와 스토리지 작업의 차이도 없어집니다.

Amazon Rekognition APIs를 데모하려면 AWS 콘솔에서 Rekognition 시도를 다루는 3단계: AWS CLI 및 AWS SDK API 사용 시작하기를 참조하세요.

주제

- Rekognition의 분석 유형 이해
- Rekognition의 이미지 및 비디오 작업 이해
- 비스토리지 및 스토리지 API 작업 이해
- 모델 버전 관리 이해

Rekognition의 분석 유형 이해

Amazon Rekognition Image API와 Amazon Rekognition Video API가 수행할 수 있는 분석 유형은 다음과 같습니다. API에 대한 정보는 Rekognition의 이미지 및 비디오 작업 이해 단원을 참조하십시오.

다음 표에는 작업 미디어 유형 및 사용 사례와 관련하여 사용해야 하는 작업이 나열되어 있습니다.

사용 사례	미디어 유형	운영
콘텐츠 조절	이미지	DetectModerationLabels, StartMediaAnalysisJob, GetMediaAnalysisJob, ListMediaAnalysisJobs
	저장된 비디오	StartContentModeration, GetContentModeration
신원 확인	이미지	CreateCollection, CreateUser, IndexFaces, AssociateFaces, SearchFacesByImage, SearchUsersByImage
	저장된 비디오	CreateCollection, IndexFaces, StartFaceSearch, GetFaceSearch
	스트리밍 비디오(얼굴 생체 확인 감지)	CreateFaceLivenessSession, StartFaceLivenessSession, GetFaceLivenessSessionResults,
얼굴 분석	이미지	DetectFaces, CompareFaces
	저장된 비디오	StartFaceDetection, GetFaceDetection
	스트리밍 비디오	CreateStreamProcessor, StartStreamProcessor
객체 및 활동 인식	이미지	DetectLabels
	저장된 비디오	StartLabelDetection, GetLabelDetection
연결된 홈	스트리밍 비디오	StartStreamProcessor

사용 사례	미디어 유형	운영
Media Analysis	저장된 비디오	StartSegmentDetection, GetSegmentDetection
작업장 안전	이미지	DetectProtectiveEquipment
텍스트 감지	이미지	DetectText
	비디오	StartTextDetection, GetTextDetection
인물 경로 추적	비디오	StartPersonTracking, GetPersonTracking
유명 인사 인식	이미지	RecognizeCelebrities
	비디오	StartCelebrityRecognition, GetCelebrityRecognition
사용자 지정 레이블 감지	이미지	DetectCustomLabels
	모델 학습	사용자 지정 레이블 개발자 안내서 참조

레이블

레이블은 객체(예: 꽃, 나무 또는 테이블), 이벤트(예: 결혼식, 졸업식, 생일 파티), 개념(예: 풍경, 저녁, 자연), 활동(예: 달리기, 농구하기) 중 어느 것이든 지칭할 수 있습니다. Amazon Rekognition은 이미지와 비디오에서 레이블을 감지할 수 있습니다. 자세한 내용은 객체 및 개념 감지 단원을 참조하십시오.

Rekognition은 이미지 및 저장된 비디오에서 대량의 레이블 목록을 감지할 수 있습니다. 또한 Rekognition은 스트리밍 비디오에서 소수의 레이블을 감지할 수도 있습니다.

다음 작업을 사용하여 자체 사용 사례를 기반으로 레이블을 감지하세요.

- 이미지에서 레이블을 감지하려면: DetectLabels를 사용합니다. 주 이미지 색상 및 이미지 품질과 같은 이미지 속성을 식별할 수 있습니다. 이를 위해 DetectLabels를 `IMAGE_PROPERTIES`와 함께 입력 파라미터로 사용하세요.

- 저장된 동영상에서 레이블을 감지하려면: StartLabelDetection을 사용합니다. 저장된 비디오에서는 주 이미지 색상 및 이미지 품질 감지가 지원되지 않습니다.
- 스트리밍 비디오에서 레이블을 감지하려면: CreateStreamProcessor를 사용하세요. 스트리밍 비디오에서는 주 이미지 색상 및 이미지 품질 감지가 지원되지 않습니다.

포함 및 제외 필터링 옵션을 사용하여 이미지 및 저장된 비디오 레이블 감지 둘 다에 대해 반환을 원하는 레이블 유형을 지정할 수 있습니다.

사용자 지정 레이블

Amazon Rekognition Custom Labels는 기계 학습 모델을 학습시켜 이미지에서 여러분의 비즈니스에 필요한 객체와 장면을 식별할 수 있습니다. 예를 들어, 로고를 감지하거나 조립 라인에서 엔지니어링 기계 부품을 감지하도록 모델을 교육할 수 있습니다.

> **Note**
>
> Amazon Rekognition Custom Labels에 대한 자세한 내용은 Amazon Rekognition Custom Labels 개발자 안내서를 참조하세요.

Amazon Rekognition에서는 기계 학습 모델을 생성, 훈련, 평가 및 실행하는 데 사용되는 콘솔을 제공합니다. 자세한 내용은 Amazon Rekognition Custom Labels 개발자 안내서의 Amazon Rekognition Custom Labels 시작을 참조하세요. Amazon Rekognition Custom Labels API를 사용하여 모델을 훈련하고 실행할 수도 있습니다. 자세한 내용은 Amazon Rekognition Custom Labels 개발자 안내서의 Amazon Rekognition Custom Labels SDK 시작을 참조하세요.

교육된 모델을 사용하여 이미지를 분석하려면 DetectCustomLabels를 사용합니다.

Face Liveness 감지

Amazon Rekognition Face Liveness를 사용하면 얼굴 기반 신원 확인을 받고 있는 사용자가 카메라 앞에 실제로 존재하며 사용자의 얼굴을 도용하는 악의적인 행위자가 아닌지 확인할 수 있습니다. 이는 카메라에 가해지는 스푸핑 공격 및 카메라를 우회하려는 공격을 탐지합니다. 사용자는 짧은 동영상 셀카를 촬영하여 Face Liveness 검사를 완료할 수 있으며, 검사 시 생체 확인 점수가 반환됩니다. 얼굴 생체 확인의 결과는 확률적 계산을 통해 결정되며 검사 후 신뢰도 점수(0~100 사이)가 반환됩니다. 점수가 높을수록 검사를 받는 사람이 실제로 존재한다는 신뢰도가 높아집니다.

Face Liveness에 대한 자세한 내용은 얼굴 생체 확인 감지 섹션을 참조하세요.

얼굴 감지 및 분석

Amazon Rekognition은 이미지와 저장된 비디오에서 얼굴을 감지할 수 있습니다. Amazon Rekognition을 사용하면 다음과 같은 정보를 얻을 수 있습니다.

- 이미지 또는 동영상에서 얼굴이 감지되는 위치
- 눈의 위치와 같은 얼굴 표식
- 이미지에 얼굴 가려짐이 있는지 여부
- 감지된 감정(예: 행복함, 슬픔)
- 이미지에서 사람의 시선이 향하는 방향

성별이나 연령과 같은 인구통계학적 정보도 해석할 수 있습니다. 이미지 속의 얼굴을 다른 이미지에서 감지된 얼굴과 비교할 수도 있습니다. 얼굴에 대한 정보도 저장하여 다중에 검색할 수 있습니다. 자세한 내용은 얼굴 감지 및 분석 단원을 참조하십시오.

이미지에서 얼굴을 감지하려면 DetectFaces를 사용합니다. 저장된 비디오에서 얼굴을 감지하려면 StartFaceDetection을 사용합니다.

얼굴 검색

Amazon Rekognition은 얼굴을 검색할 수 있습니다. 얼굴 정보는 모음이라고 하는 컨테이너로 인덱싱됩니다. 그런 다음 모음의 얼굴 정보는 이미지, 저장된 비디오, 스트리밍 비디오에서 감지된 얼굴과 일치시킬 수 있습니다. 자세한 설명은 컬렉션에서 얼굴 검색 섹션을 참조하세요.

이미지에서 알려진 얼굴을 감지하려면 DetectFaces를 사용합니다. 저장된 비디오에서 알려진 얼굴을 감지하려면 StartFaceDetection을 사용합니다. 스트리밍 비디오에서 알려진 얼굴을 감지하려면 CreateStreamProcessor를 사용합니다.

인물 경로

Amazon Rekognition Video로 저장된 비디오에서 감지된 인물의 경로를 추적할 수 있습니다. Amazon Rekognition Video는 비디오에서 감지된 인물에 대한 경로 추적, 얼굴 세부 정보 및 프레임 내 위치 정보를 제공합니다. 자세한 내용은 인물 경로 추적 단원을 참조하십시오.

저장된 비디오에서 사람을 감지하려면 StartPersonTracking을 사용합니다.

개인용 보호 장비

Amazon Rekognition은 이미지에서 감지된 사람이 착용한 PPE(개인 보호 장비)를 감지할 수 있습니다. Amazon Rekognition은 손 덮개, 얼굴 덮개 및 머리 덮개를 감지합니다. Amazon Rekognition은 PPE 물품이 적절한 신체 부위를 덮고 있는지 예측합니다. 감지된 사람 및 PPE 물품의 경계 상자를 확인할 수도 있습니다. 자세한 내용은 개인 보호 장비 감지 단원을 참조하십시오.

이미지에서 PPE를 감지하려면 DetectProtectiveEquipment를 사용합니다.

유명 인사

Amazon Rekognition은 이미지와 저장된 비디오에서 수천 명의 유명 인사를 인식할 수 있습니다. 이미지에서 유명 인사 얼굴이 감지된 부분, 얼굴 표식 및 유명 인사의 얼굴 표정에 대한 정보를 가져올 수 있습니다. 저장된 비디오 전체에 나타나는 유명 인사에 대한 추적 정보를 가져올 수 있습니다. 또한 감정 표현, 성별 표현 등 인식된 유명 인사에 대한 추가 정보도 얻을 수 있습니다. 자세한 내용은 유명 인사 인식 단원을 참조하십시오.

이미지에서 유명 인사를 인식하려면 RecognizeCelebrities를 사용합니다. 저장된 비디오에서 유명 인사를 인식하려면 StartCelebrityRecognition을 사용합니다.

텍스트 감지

Amazon Rekognition Text in Image는 이미지의 텍스트를 감지하여 기계 판독 가능한 텍스트로 변환합니다. 자세한 내용은 텍스트 감지 단원을 참조하십시오.

이미지에서 텍스트를 감지하려면 DetectText를 사용합니다.

부적절하거나 불쾌감을 주는 콘텐츠

Amazon Rekognition은 이미지와 저장된 비디오를 분석하여 성인 및 폭력적인 콘텐츠를 찾아낼 수 있습니다. 자세한 내용은 콘텐츠 조절 단원을 참조하십시오.

안전하지 않은 이미지를 감지하려면 DetectModerationLabels를 사용합니다. 안전하지 않은 저장된 비디오를 감지하려면 StartContentModeration을 사용합니다.

사용자 지정

Rekognition에서 제공하는 특정 이미지 분석 API를 사용하면 자체 데이터를 기반으로 학습된 사용자 지정 어댑터를 만들어 딥 러닝 모델의 정확도를 높일 수 있습니다. 어댑터는 Rekognition의 사전 학습된 딥 러닝 모델에 플러그인하는 구성 요소로, 자체 이미지에 기반한 도메인 지식을 통해 정확도를 향상시킵니다. 샘플 이미지를 제공하고 주석을 달아 필요에 맞게 어댑터를 학습시킬 수 있습니다.

어댑터를 생성하면 AdapterId가 제공됩니다. 작업에 이 AdapterId를 제공하여 생성한 어댑터를 사용하도록 지정할 수 있습니다. 예를 들어, 동기식 이미지 분석을 위해 DetectModerationLabels API에 AdapterId를 제공합니다. 요청의 일부로 AdapterId를 제공하면 Rekognition에서 자동으로 이를 사용하여 이미지에 대한 예측을 개선합니다. 이를 통해 Rekognition의 기능을 활용하면서 필요에 맞게 사용자 지정할 수 있습니다.

또한 StartMediaAnalysisJob API를 사용하여 이미지에 대한 예측을 대량으로 얻을 수 있는 옵션도 있습니다. 자세한 내용은 대량 분석을 참조하세요.

Rekognition 콘솔에 이미지를 업로드하고 이러한 이미지에 대한 분석을 실행하여 Rekognition 작업의 정확성을 평가할 수 있습니다. Rekognition이 선택한 기능을 사용하여 이미지에 주석을 달고 나면 사용자는 검증된 예측을 사용하여 예측을 검토하고 어댑터를 생성하는 것이 유용할 레이블이 어떤 것인지 결정할 수 있습니다.

현재는 DetectModerationLabels를 어댑터와 사용할 수 있습니다. 어댑터 생성 및 사용에 대한 자세한 내용은 사용자 지정 조절을 통한 정확도 향상 섹션을 참조하세요.

대량 분석

Rekognition 대량 분석을 사용하면 StartMediaAnalysisJob 작업과 더불어 매니페스트 파일을 사용하여 대규모 이미지 컬렉션을 비동기적으로 처리할 수 있습니다. 자세한 내용은 대량 분석을 참조하세요.

Rekognition의 이미지 및 비디오 작업 이해

Amazon Rekognition은 이미지 및 비디오 분석을 위한 두 개의 기본 API 세트를 제공합니다.

- Amazon Rekognition Image: 이 API는 이미지 분석을 위해 설계되었습니다.
- Amazon Rekognition Video: 이 API는 저장된 비디오와 스트리밍 비디오를 모두 분석하는 데 중점을 둡니다.

두 API 모두 얼굴 및 객체와 같은 다양한 개체를 감지할 수 있습니다. 지원되는 비교 및 감지 유형을 포괄적으로 이해하려면 Rekognition의 분석 유형 이해의 섹션을 참조하세요.

Amazon Rekognition Image 작업

Amazon Rekognition Image 작업은 동기식입니다. 입력과 응답은 JSON 형식입니다. Amazon Rekognition Image 작업은 .jpg나 .png 이미지 형식의 입력 이미지를 분석합니다. Amazon Rekognition Image 작업에 전달된 이미지는 Amazon S3 버킷에 저장할 수 있습니다. AWS CLI를 사용하지 않는 경

우 Base64 인코딩 이미지 바이트를 Amazon Rekognition 작업에 직접 전달할 수도 있습니다. 자세한 내용은 이미지 작업을 참조하세요.

Amazon Rekognition Video 작업

Amazon Rekognition Video API는 Amazon S3 버킷에 저장되거나 Amazon Kinesis Video Streams를 통해 스트리밍되는 비디오의 분석을 용이하게 합니다.

저장된 비디오 작업의 경우 다음을 참고하세요.

- 작업은 비동기식으로 수행됩니다.
- 분석은 “Start” 작업(예: 저장된 비디오 속 얼굴 감지의 경우 StartFaceDetection)으로 시작해야 합니다.
- 분석의 완료 상태는 Amazon SNS 주제에 게시됩니다.
- 분석 결과를 검색하려면 해당 “Get” 작업(예: GetFaceDetection)을 사용합니다.
- 자세한 내용은 저장된 비디오 분석 작업을 참조하세요.

스트리밍 비디오 분석의 경우:

- Rekognition Video 컬렉션 및 레이블(객체 또는 개념) 감지에 얼굴 검색이 포함되는 기능입니다.
- 레이블에 대한 분석 결과는 Amazon SNS 및 Amazon S3 알림으로 전송됩니다.
- 얼굴 검색 결과가 Kinesis 데이터 스트림으로 출력됩니다.
- 스트리밍 비디오 분석 관리는 Amazon Rekognition Video 스트림 프로세서를 통해 수행됩니다(예: CreateStreamProcessor를 사용하여 프로세서 생성).
- 자세한 내용은 스트리밍 비디오 이벤트 작업을 참조하세요.

각 비디오 분석 작업은 분석 중인 비디오에 대한 메타데이터와 작업 ID 및 작업 태그를 반환합니다. 레이블 감지 및 비디오 콘텐츠 조절과 같은 작업을 통해 타임스탬프 또는 레이블 이름을 기준으로 정렬하고 타임스탬프 또는 비디오 세그먼트별로 결과를 집계할 수 있습니다.

비스토리지 및 스토리지 기반 작업

Amazon Rekognition 작업은 다음과 같은 카테고리로 그룹화됩니다.

- 비스토리지 API 작업 - 이 작업의 경우 Amazon Rekognition은 어떤 정보도 유지하지 않습니다. 여러분이 입력 이미지와 비디오를 제공하면 작업이 분석을 수행하고 결과를 반환하지만 Amazon

Rekognition에는 아무것도 저장되지 않습니다. 자세한 내용은 비스토리지 작업 단원을 참조하십시오.

- 스토리지 기반 API 작업 - Amazon Rekognition 서버는 탐지된 얼굴 정보를 컬렉션이라고 하는 컨테이너에 저장할 수 있습니다. Amazon Rekognition은 유지되는 얼굴 정보에서 얼굴 일치를 검색하는 데 사용할 수 있는 추가 API 작업을 제공합니다. 자세한 내용은 스토리지 기반 API 작업 단원을 참조하십시오.

AWS SDK 또는 HTTP를 사용하여 Amazon Rekognition API 작업 직접 호출

AWS SDK를 사용하거나 직접 HTTP를 사용하여 Amazon Rekognition API 작업을 직접 호출할 수 있습니다. 특별한 이유가 없다면 항상 AWS SDK를 사용해야 합니다. 이 단원의 Java 예제는 AWS SDK를 사용합니다. Java 프로젝트 파일은 제공되지 않지만 AWS Toolkit for Eclipse를 사용하여 Java를 사용하는 AWS 애플리케이션을 개발할 수 있습니다.

이 단원의 .NET 예제는 AWS SDK for .NET을 사용합니다. AWS Toolkit for Visual Studio을 사용하면 .NET을 사용하는 AWS 애플리케이션을 개발할 수 있습니다. 여기에는 애플리케이션 배포 및 서비스 관리를 위한 AWS Explorer와 유용한 템플릿이 포함됩니다.

이 안내서의 API 참조에서는 HTTP를 사용한 Amazon Rekognition 작업 직접 호출을 다룹니다. Java 참조 정보는 AWS SDK for Java를 참조하십시오.

사용 가능한 Amazon Rekognition 서비스 엔드포인트는 AWS 리전 및 엔드포인트에 나와 있습니다.

HTTP를 사용하여 Amazon Rekognition을 직접 호출할 때는 POST HTTP 작업을 사용합니다.

비스토리지 및 스토리지 API 작업 이해

Amazon Rekognition은 두 가지 유형의 API 작업을 제공합니다. 즉 Amazon Rekognition에 어떤 정보도 저장되지 않는 비스토리지 작업과 특정 얼굴 정보가 Amazon Rekognition에 저장되는 스토리지 작업입니다.

비스토리지 작업

Amazon Rekognition은 이미지를 대상으로 다음과 같은 비스토리지 API 작업을 제공합니다.

- DetectLabels
- DetectFaces

- CompareFaces
- DetectModerationLabels
- DetectProtectiveEquipment
- RecognizeCelebrities
- DetectText
- GetCelebrityInfo

Amazon Rekognition은 비디오를 대상으로 다음과 같은 비스토리지 API 작업을 제공합니다.

- StartLabelDetection
- StartFaceDetection
- StartPersonTracking
- StartCelebrityRecognition
- StartContentModeration

이러한 작업을 비스토리지 API 작업이라고 하는 이유는 작업을 직접 호출할 때 Amazon Rekognition이 입력 이미지에 관해 발견된 어떤 정보도 유지하지 않기 때문입니다. 다른 모든 Amazon Rekognition API 작업과 마찬가지로 비스토리지 API 작업은 입력 이미지 바이트를 유지하지 않습니다.

다음 예제 시나리오는 애플리케이션에 비 스토리지 API 작업을 통합할 수 있는 경우를 보여 줍니다. 이 시나리오에서는 로컬 이미지 리포지토리가 있다고 가정합니다.

Example 1: 로컬 리포지토리에서 특정 레이블이 포함된 이미지를 찾는 애플리케이션

먼저 리포지토리에 있는 각 이미지에서 Amazon Rekognition `DetectLabels` 작업을 사용하여 레이블(객체 및 개념)을 감지하고 다음과 같이 클라이언트 측 인덱스를 빌드합니다.

```
Label        ImageID

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

그러면 애플리케이션이 이 인덱스를 검색해 로컬 리포지토리에서 특정 레이블이 포함된 이미지를 찾을 수 있습니다. 예를 들어 나무가 포함된 이미지를 표시하십시오.

Amazon Rekognition이 감지하는 각 레이블에는 연결된 신뢰도 값이 있습니다. 신뢰도 값은 입력 이미지에 해당 레이블이 포함될 신뢰도 수준을 나타냅니다. 이 신뢰도 값을 사용하여 필요하다면 감지의 신뢰도 수준에 대한 애플리케이션 요구 사항에 따라 레이블에서 추가적인 클라이언트 측 필터링을 수행할 수 있습니다. 예를 들어 정확한 레이블이 필요하다면 신뢰도가 더 높은(95% 이상) 레이블만을 필터링하고 선택할 수 있습니다. 애플리케이션에 이보다 높은 신뢰도 값이 필요하지 않은 경우에는 신뢰도 값이 보다 낮은(50% 정도) 레이블을 필터링할 수 있습니다.

Example 2: 향상된 얼굴 이미지를 표시하는 애플리케이션

먼저 Amazon Rekognition `DetectFaces` 작업을 사용하여 로컬 리포지토리에 있는 각 이미지에서 얼굴을 탐지하고 클라이언트 측 인덱스를 빌드할 수 있습니다. 이 작업은 각 얼굴에 대해 경계 상자, 얼굴 표식(예: 입과 귀의 위치), 얼굴 속성(예: 성별)이 포함된 메타데이터를 반환합니다. 이 메타데이터는 다음과 같이 클라이언트 측 로컬 인덱스에 저장할 수 있습니다.

```
ImageID     FaceID     FaceMetaData

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

이 인덱스에서 기본 키는 `ImageID`와 `FaceID`의 조합입니다.

그런 다음 애플리케이션이 로컬 리포지토리에서 이미지를 표시할 때 인덱스의 정보를 사용하여 이미지를 향상시킬 수 있습니다. 예를 들어 얼굴 주위에 경계 상자를 추가하거나 얼굴 특징을 강조 표시할 수 있습니다.

스토리지 기반 API 작업

Amazon Rekognition Image는 이미지에서 얼굴을 감지하고 Amazon Rekognition 컬렉션에서 감지된 얼굴 특성에 대한 정보를 유지하는 데 사용할 수 있는 IndexFaces 작업을 지원합니다. 이것이 스토리지 기반 API 작업의 예인 이유는 서비스가 서버에서 정보를 유지하기 때문입니다.

Amazon Rekognition Image는 다음과 같은 스토리지 API 작업을 제공합니다.

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

Amazon Rekognition Video는 다음과 같은 스토리지 API 작업을 제공합니다.

- StartFaceSearch
- CreateStreamProcessor

얼굴 정보를 저장하려면 먼저 계정의 AWS 리전 중 하나에서 얼굴 모음을 만들어야 합니다. `IndexFaces` 작업을 호출할 때 이 얼굴 모음을 지정합니다. 얼굴 모음을 만들고 모든 얼굴의 얼굴 특징 정보를 저장한 후에 모음에서 얼굴 일치를 검색할 수 있습니다. 예를 들어 이미지에서 가장 큰 얼굴을 감지하고 모음에서 일치하는 얼굴을 감지하려면 `searchFacesByImage.`를 호출합니다.

`IndexFaces`를 통해 컬렉션에 저장된 얼굴 정보는 Amazon Rekognition Video 작업에 액세스할 수 있습니다. 예를 들어, StartFaceSearch 직접 호출을 통해 기존 컬렉션의 얼굴과 일치하는 사람의 비디오를 검색할 수 있습니다.

모음을 만들고 관리하는 방법에 대한 정보는 컬렉션에서 얼굴 검색를 참조하십시오.

> **Note**
>
> 컬렉션에는 얼굴의 수학적 표현인 얼굴 벡터가 저장됩니다. 컬렉션에는 얼굴 이미지가 저장되지 않습니다.

Example 1: 건물 액세스를 인증하는 애플리케이션

먼저 얼굴을 추출해 검색 가능한 이미지 벡터로 저장하는 `IndexFaces` 작업을 사용하여 얼굴 모음을 만들어 스캔한 배지 이미지를 저장할 수 있습니다. 그런 다음 직원이 건물에 들어오면 직원 얼굴 이미

지가 캡처되어 SearchFacesByImage 작업으로 전송됩니다. 얼굴 일치에서 충분히 높은 유사성 점수(가령 99%)가 나오면 직원을 인증할 수 있습니다.

모델 버전 관리 이해

Amazon Rekognition은 딥 러닝 모델을 사용하여 얼굴 감지를 수행하고 컬렉션에서 얼굴을 감지합니다. 그리고 계속해서 고객 피드백과 딥 러닝 연구 발전을 토대로 모델 정확도를 개선하고 있습니다. 이러한 개선 사항은 모델 업데이트로 전달됩니다. 예를 들어 모델 버전 1.0에서는 IndexFaces가 이미지에서 가장 큰 얼굴 15개를 인덱싱할 수 있습니다. 최신 버전의 모델을 통해 IndexFaces는 이미지에서 가장 큰 얼굴 100개를 인덱싱할 수 있습니다.

새 모음을 만들 때 최신 버전의 모델과 연결됩니다. 정확도를 개선하기 위해 모델은 정기적으로 업데이트됩니다.

새 버전의 모델이 릴리스되면 다음 상황이 나타납니다.

- 만든 새 모음은 최신 모델과 연결됩니다. IndexFaces를 사용하여 새 컬렉션에 추가하는 얼굴은 최신 모델을 사용하여 감지됩니다.
- 기존 모음은 모음을 만들 때 사용했던 모델 버전을 계속 사용합니다. 이 모음에 저장된 얼굴 벡터는 최신 모델 버전으로 자동 업데이트되지 않습니다.
- 기존 모음에 추가되는 새 얼굴은 이미 모음에 연결된 모델을 사용하여 감지합니다.

다른 모델 버전과는 호환되지 않습니다. 특히 서로 다른 모델 버전을 사용하는 여러 모음으로 이미지를 인덱싱하는 경우 감지된 동일한 얼굴의 얼굴 식별자는 서로 다릅니다. 동일한 모델과 연결된 여러 모음으로 이미지가 인덱싱되는 경우 얼굴 식별자는 동일합니다.

모음 관리에서 모델 업데이트를 고려하지 않을 경우 애플리케이션이 호환성 문제를 겪을 수 있습니다. 모음 작업(예: CreateCollection)의 응답으로 반환된 FaceModelVersion 필드를 사용하여 모음이 사용하는 모델의 버전을 확인할 수 있습니다. DescribeCollection 직접 호출을 통해 기존 컬렉션의 모델 버전을 가져올 수도 있습니다. 자세한 내용은 컬렉션 설명 단원을 참조하십시오.

모음에 있는 기존 얼굴 벡터는 최신 모델 버전으로 자동 업데이트되지 않습니다. Amazon Rekognition은 소스 이미지 바이트를 저장하지 않기 때문에 최신 버전의 모델을 사용하여 이미지를 자동으로 다시 인덱싱할 수 없습니다.

기존 컬렉션에 저장된 얼굴에서 최신 모델을 사용하려면 새 컬렉션(CreateCollection)을 만들고 소스 이미지를 새 컬렉션(Indexfaces)으로 다시 인덱싱합니다. 새 모음의 얼굴 식별자는 이전 모음의 얼굴

식별자와 다르기 때문에 사용 중인 애플리케이션에서 저장한 모든 얼굴 식별자를 업데이트해야 합니다. 이전 컬렉션이 더 이상 필요하지 않으면 DeleteCollection을 사용하여 삭제할 수 있습니다.

DetectFaces와 같은 무상태 작업은 모델의 최신 버전을 사용합니다.

Amazon Rekognition 시작

이 섹션에서는 Amazon Rekognition 사용 시작과 관련된 주제를 다룹니다. Amazon Rekognition을 처음 사용하는 경우, 먼저 Amazon Rekognition 작동 방식에 나와 있는 개념과 용어를 검토하는 것이 좋습니다.

Rekognition을 사용하려면 먼저 AWS 계정을 생성하고 AWS 계정 ID를 얻어야 합니다. 또한 Amazon Rekognition 시스템에서 리소스에 액세스하는 데 필요한 권한이 있는지 확인할 수 있도록 사용자를 생성해야 합니다.

계정을 생성한 후 및 AWS CLI SDK를 AWS 설치하고 구성해야 합니다. SDKs AWS CLI 를 사용하면 명령줄을 통해 Amazon Rekognition 및 기타 서비스와 상호 작용할 수 있고, AWS SDKs를 사용하면 Java 및 Python과 같은 프로그래밍 언어를 사용하여 Amazon Rekognition과 상호 작용할 수 있습니다.

AWS CLI 및 AWS SDKs를 설정한 후에는 두 가지를 모두 사용하는 방법의 몇 가지 예를 살펴볼 수 있습니다. 또한 콘솔을 사용하여 Amazon Rekognition과 상호 작용하는 방법에 대한 몇 가지 예를 볼 수 있습니다.

Amazon Rekognition 콘솔을 통해 대량 분석 및 사용자 지정 조절 워크플로를 사용할 수 있습니다. 이러한 기능 외에도 콘솔을 사용하여 다양한 Amazon Rekognition 기능을 시연할 수도 있습니다. Amazon Rekognition 기능을 최대한 활용하려면 API를 사용해야 합니다.

주제

- 1단계: AWS 계정 설정 및 사용자 생성
- 2단계: AWS CLI 및 AWS SDKs 설정
- 3단계: AWS CLI 및 AWS SDK API 사용 시작하기
- 4단계: Amazon Rekognition 콘솔을 사용하여 시작

1단계: AWS 계정 설정 및 사용자 생성

Amazon Rekognition을 처음 사용하기 전에 다음 작업을 완료해야 합니다.

1. AWS 계정에 가입합니다.
2. 사용자를 생성합니다.

개발자 안내서의 이 섹션에서는 AWS 계정과 사용자를 생성하는 이유와 방법을 설명합니다.

주제

- AWS 계정 및 사용자 생성

AWS 계정 및 사용자 생성

AWS 계정

Amazon Web Services(AWS)에 가입하면 Amazon Rekognition을 포함해 AWS의 모든 서비스에 AWS 계정이 자동으로 등록됩니다. 사용한 서비스에 대해서만 청구됩니다.

Amazon Rekognition에서는 사용한 리소스에 대해서만 비용을 지불합니다.

신규 AWS 고객인 경우 Amazon Rekognition을 무료로 시작할 수 있습니다. 자세한 내용은 AWS 프리 티어를 참조하십시오.

계정 생성 지침은 다음 에 가입 AWS 계정 섹션을 참조하세요.

이미 AWS 계정이 있는 경우 계정 설정을 건너뛰고 관리 사용자를 생성합니다.

Users

Amazon Rekognition과 같은 AWS 서비스를 사용하려면 액세스할 때 보안 인증 정보를 제공해야 합니다. 이를 통해 서비스가 소유한 리소스에 액세스할 수 있는 권한이 있는지를 확인합니다.

콘솔을 사용하려면 암호가 필요한 동안 AWS 계정에 대한 액세스 키를 생성하여 AWS CLI 또는 APIs에 액세스할 수 있습니다. 그러나 AWS 계정 루트 사용자의 자격 증명을 사용하여 AWS에 액세스하는 것은 권장되지 않습니다. 대신 AWS Identity and Access Management (IAM)을 사용하여 관리 사용자를 생성하는 것이 좋습니다.

그러면 특별 URL과 해당 관리자의 보안 인증 정보를 사용하여 AWS에 액세스할 수 있습니다.

AWS에 가입했지만 아직 사용자를 생성하지 않았다면, IAM 콘솔을 사용하여 사용자를 생성할 수 있습니다. 관리자 생성 방법에 대한 지침은 다음 관리자 액세스 권한이 있는 사용자 생성 섹션을 참조하세요.

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 확인하고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 그룹 추가를 참조하세요.

2단계: AWS CLI 및 AWS SDKs 설정

주제

- 프로그래밍 방식 액세스 권한 부여
- AWS SDK에서 Rekognition 사용

다음 단계에서는이 설명서의 예제에서 사용하는 AWS Command Line Interface (AWS CLI) 및 AWS SDKs를 설치하는 방법을 보여줍니다. AWS SDK 호출을 인증하는 방법에는 여러 가지가 있습니다. 이 가이드의 예제에서는 AWS CLI 명령 및 AWS SDK API 작업을 호출하기 위해 기본 자격 증명 프로파일을 사용하고 있다고 가정합니다.

사용 가능한 AWS 리전 목록은의 리전 및 엔드포인트를 참조하세요Amazon Web Services 일반 참조.

단계에 따라 AWS SDKs를 다운로드하고 구성합니다.

AWS CLI 및 AWS SDKs를 설정하려면

1. 및 사용하려는 AWS SDKs를 다운로드AWS CLI하여 설치합니다. 이 가이드에서는 Java AWS CLI, Python, Ruby, Node.js, PHP, .NET 및 JavaScript에 대한 예제를 제공합니다. SDK 설치 AWS SDKs. https://aws.amazon.com/tools/
2. AWS 계정 및 사용자 생성에서 만든 사용자의 액세스 키를 만듭니다.

   a. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.

   b. 탐색 창에서 사용자를 선택합니다.

   c. AWS 계정 및 사용자 생성에서 만든 사용자의 이름을 선택합니다.

   d. Security credentials(보안 자격 증명) 탭을 선택합니다.

   e. 액세스 키 생성을 선택합니다. 그런 다음 .csv 파일 다운로드를 선택하여 액세스 키 ID 및 보안 액세스 키를 컴퓨터에 CSV 파일로 저장합니다. 안전한 위치에 파일을 저장합니다. 이 대화 상자를 닫은 후에는 비밀 액세스 키에 다시 액세스할 수 없습니다. CSV 파일을 다운로드한 후 [Close]를 클릭합니다.
3. 를 설치한 경우 명령 프롬프트에를 입력하여 대부분의SDK에 대한 자격 증명과 리전을 구성할 AWS CLI수 있습니다. AWS SDKs `aws configure` 그렇지 않은 경우 다음 지침을 따릅니다.
4. 컴퓨터에서 홈 디렉터리를 탐색하고 `.aws` 디렉터리를 생성합니다. Linux 또는 macOS 같은 Unix 기반 시스템의 경우 이 디렉터리의 위치는 다음과 같습니다.

   ```
   ~/.aws
   ```

   Windows에서 이 디렉터리의 위치는 다음과 같습니다.

   ```
   %HOMEPATH%\.aws
   ```
5. `.aws` 디렉터리에서 `credentials`라는 파일을 새로 생성합니다.
6. 2단계에서 만든 자격 증명 CSV 파일을 열어서 다음 형식을 사용하는 `credentials` 파일로 내용을 복사합니다.

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

your_access_key_id 및 your_secret_access_key를 사용자의 액세스 키 ID와 비밀 액세스 키로 대체합니다.

7. `Credentials` 파일을 저장하고 CSV 파일을 삭제합니다.
8. `.aws` 디렉터리에서 `config`라는 파일을 새로 생성합니다.
9. `config` 파일을 열고 리전을 다음 형식으로 입력합니다.

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

your_aws_region을 원하는 AWS 리전(예를 들어 `us-west-2`)으로 대체합니다.

> **Note**
>
> 리전을 선택하지 않으면 기본적으로 us-east-1이 사용됩니다.

10. `config` 파일을 저장합니다.

프로그래밍 방식 액세스 권한 부여

이 가이드의 AWS CLI 및 코드 예제는 로컬 컴퓨터 또는 Amazon Elastic Compute Cloud 인스턴스와 같은 기타 AWS 환경에서 실행할 수 있습니다. 예제를 실행하려면 예제에서 사용하는 AWS SDK 작업에 대한 액세스 권한을 부여해야 합니다.

주제

- 로컬 컴퓨터에서 코드 실행
- AWS 환경에서 코드 실행

로컬 컴퓨터에서 코드 실행

로컬 컴퓨터에서 코드를 실행하려면 단기 자격 증명을 사용하여 사용자에게 AWS SDK 작업에 대한 액세스 권한을 부여하는 것이 좋습니다. 로컬 컴퓨터에서 AWS CLI 및 코드 예제를 실행하는 방법에 대한 자세한 내용은 섹션을 참조하세요로컬 컴퓨터에서 프로필 사용.

사용자는 AWS 외부에서와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다 AWS Management Console. 프로그래밍 방식 액세스를 부여하는 방법은에 액세스하는 사용자 유형에 따라 다릅니다 AWS.

사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

프로그래밍 방식 액세스가 필요한 사용자	목적	방법
IAM	(권장) 콘솔 자격 증명을 임시 자격 증명으로 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 AWS 로컬 개발을 위한 로그인을 참조하세요. • AWS SDKs 경우 SDK 및 도구 참조 안내서의 AWS 로컬 개발을 위한 로그인을 참조하세요. AWS SDKs
작업 인력 ID (IAM Identity Center에서 관리되는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 를 사용하도록 구성을 AWS IAM Identity Center 참조하세요. • AWS SDKs, 도구 및 AWS APIs의 경우 SDK 및 도구 참조 안내서의 IAM Identity Center 인증을 참조하세요. AWS SDKs
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	IAM 사용 설명서의 AWS 리소스에서 임시 자격 증명 사용의 지침을 따릅니다.

프로그래밍 방식 액세스가 필요한 사용자	목적	방법
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 IAM 사용자 자격 증명을 사용하여 인증을 참조하세요. • AWS SDKs 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용하여 인증을 참조하세요. AWS SDKs • AWS APIs 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하세요.

로컬 컴퓨터에서 프로필 사용

에서 생성한 단기 자격 증명을 사용하여이 가이드의 AWS CLI 및 코드 예제를 실행할 수 있습니다로컬 컴퓨터에서 코드 실행. 보안 인증 정보 및 기타 설정 정보를 가져오려면, 예제에서는 `profile-name` 이름의 프로필을 사용합니다. 예를 들어:

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

프로필이 나타내는 사용자에게는 Rekognition SDK 작업 및 예제에 필요한 기타 AWS SDK 작업을 호출할 수 있는 권한이 있어야 합니다.

AWS CLI 및 코드 예제와 함께 작동하는 프로파일을 생성하려면 다음 중 하나를 선택합니다. 생성한 프로필의 이름이 `profile-name`인지 확인하세요.

- IAM으로 관리하는 사용자 — IAM 역할로 전환(AWS CLI)의 지침을 따르세요.

- 작업 인력 자격 증명(사용자 관리형 AWS IAM Identity Center) - 사용할 AWS CLI 구성의 AWS IAM Identity Center 지침을 따릅니다. 코드 예제의 경우 IAM ID 센터를 통한 인증을 지원하는 AWS 툴킷을 지원하는 통합 개발 환경(IDE)을 사용하는 것이 좋습니다. Java 예제는 Java로 구축 시작을 참조하세요. Python 예제는 Python으로 구축 시작을 참조하세요. 자세한 정보는 IAM Identity Center 보안 인증 정보를 참조하세요.

> **Note**
>
> 코드를 사용하여 단기 보안 인증 정보를 얻을 수 있습니다. 자세한 내용은 IAM 역할로 전환(API)을 참조하세요. IAM Identity Center의 경우 CLI 액세스를 위한 IAM 역할 자격 증명 가져오기의 지침에 따라 역할에 대한 단기 보안 인증 정보를 받으세요.

AWS 환경에서 코드 실행

사용자 자격 증명을 사용하여 AWS Lambda 함수에서 실행되는 프로덕션 코드와 같은 AWS 환경에서 AWS SDK 호출에 서명해서는 안 됩니다. 대신 코드에 필요한 권한을 정의하는 역할을 구성합니다. 그런 다음 코드가 실행되는 환경에 역할을 연결합니다. 역할을 연결하고 임시 보안 인증 정보를 사용할 수 있게 하는 방법은 코드가 실행되는 환경에 따라 달라집니다.

- AWS Lambda 함수 - Lambda 함수의 실행 역할을 수임할 때 Lambda가 함수에 자동으로 제공하는 임시 자격 증명을 사용합니다. 보안 인증 정보는 Lambda 환경 변수에서 사용할 수 있습니다. 프로필을 지정할 필요가 없습니다. 자세한 내용을 알아보려면 Lambda 실행 역할을 참조하세요.
- Amazon EC2 - Amazon EC2 인스턴스 메타데이터 엔드포인트 보안 인증 인증 정보를 사용합니다. 공급자는 Amazon EC2 인스턴스에 연결한 Amazon EC2 인스턴스 프로파일을 사용하여 자동으로 보안 인증 정보를 생성하고 새로 고칩니다. 자세한 내용은 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.
- Amazon Elastic 컨테이너 서비스 - 컨테이너 보안 인증 정보 공급자를 사용하세요. Amazon ECS는 메타데이터 엔드포인트로 보안 인증 정보를 전송하고 새로 고칩니다. 지정한 작업 IAM 역할은 애플리케이션이 사용하는 보안 인증 정보를 관리하기 위한 전략을 제공합니다. 자세한 내용은 AWS 서비스와 상호 작용을 참조하세요.

보안 인증 정보 공급자에 대한 자세한 정보는 표준화된 보안 인증 정보 공급자를 참조하세요.

AWS SDK에서 Rekognition 사용

AWS 소프트웨어 개발 키트(SDKs)는 널리 사용되는 많은 프로그래밍 언어에 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예제 및 설명서를 제공합니다.

SDK 설명서	코드 예제
AWS SDK for C++	AWS SDK for C++ 코드 예제
AWS CLI	AWS CLI 코드 예제
AWS SDK for Go	AWS SDK for Go 코드 예제
AWS SDK for Java	AWS SDK for Java 코드 예제
AWS SDK for JavaScript	AWS SDK for JavaScript 코드 예제
AWS SDK for Kotlin	AWS SDK for Kotlin 코드 예제
AWS SDK for .NET	AWS SDK for .NET 코드 예제
AWS SDK for PHP	AWS SDK for PHP 코드 예제
AWS Tools for PowerShell	AWS Tools for PowerShell 코드 예제
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 코드 예제
AWS SDK for Ruby	AWS SDK for Ruby 코드 예제
AWS SDK for Rust	AWS SDK for Rust 코드 예제
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP 코드 예제
AWS SDK for Swift	AWS SDK for Swift 코드 예제

Rekognition에 특화된 예제는 AWS SDKs를 사용한 Amazon Rekognition 코드 예제 섹션을 참조하세요.

> **예제 가용성**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

3단계: AWS CLI 및 AWS SDK API 사용 시작하기

사용하려는 AWS CLI 및 AWS SDKs를 설정한 후 Amazon Rekognition을 사용하는 애플리케이션을 빌드할 수 있습니다. Amazon Rekognition의 일부 기능에 콘솔 워크플로가 있지만 Amazon Rekognition과의 상호 작용은 대부분 API 작업을 사용하여 이루어집니다.

다음 주제에서는 AWS CLI 또는 AWS SDKs를 통해 Amazon Rekognition Image 및 Amazon Rekognition Video를 시작하는 방법을 보여줍니다.

- 이미지 작업 - Amazon Rekognition Image로 이미지를 분석하는 프로세스를 다룹니다.
- 저장된 비디오 분석 작업 - Amazon Rekognition Video를 사용하여 저장된 비스트리밍 비디오를 분석하는 프로세스를 다룹니다.
- 스트리밍 비디오 이벤트 작업 - Amazon Rekognition Video를 사용하여 스트리밍 비디오를 분석하는 프로세스를 다룹니다.

위에 나열된 섹션에는 AWS CLI를 사용하는 예제가 있습니다. 를 사용하려는 경우 API 호출의 형식을 지정하는 방법에 대한 자세한 내용은 다음 섹션을 AWS CLI참조하세요.

AWS CLI 예제 형식 지정

이 가이드의 AWS CLI 예제는 Linux 운영 체제용으로 형식이 지정되어 있습니다. Microsoft Windows에서 샘플을 사용하려면 `--image` 파라미터의 JSON 형식을 변경하고 줄 바꿈을 백슬래시(\)에서 캐럿 기호(^)로 변경해야 합니다. JSON 형식에 대한 자세한 내용은 AWS 명령줄 인터페이스 파라미터 값 지정을 참조하십시오.

다음은 Microsoft Windows용으로 형식이 지정된 AWS CLI 명령의 예입니다(이러한 명령은 그대로 실행되지 않으며 단지 예제의 형식일 뿐임).

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

Microsoft Windows와 Linux에서 모두 작동하는 JSON 간편 버전을 제공할 수도 있습니다.

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

자세한 내용은 AWS 명령줄 인터페이스에서 간편 구문 사용을 참조하십시오.

다음 단계

4단계: Amazon Rekognition 콘솔을 사용하여 시작

4단계: Amazon Rekognition 콘솔을 사용하여 시작

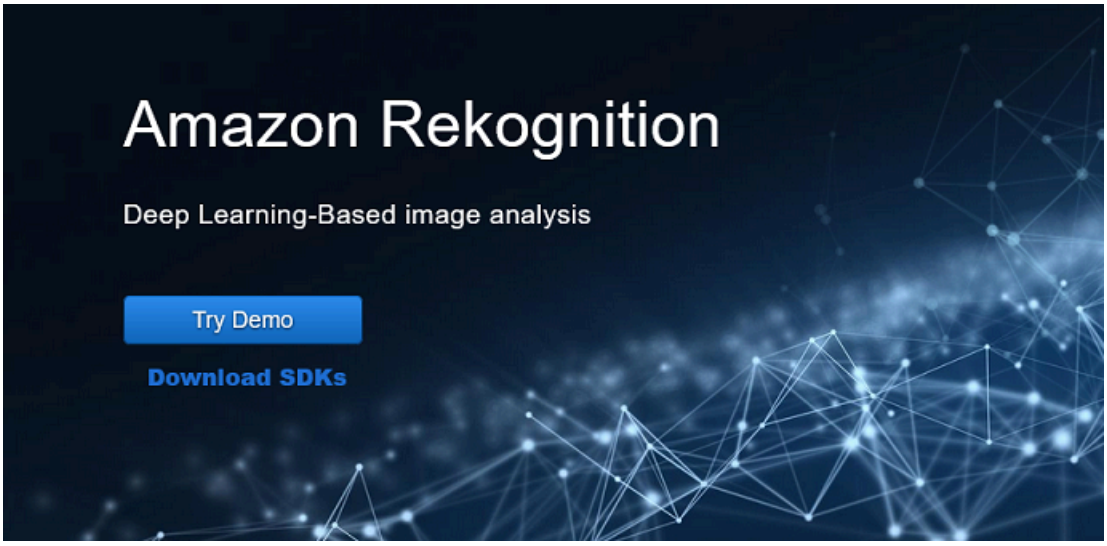
Amazon Rekognition 콘솔을 사용하면 Rekognition Custom Labels 및 Custom Moderation 기능과 관련된 리소스를 관리할 수 있습니다. 콘솔은 다른 Rekognition 기능의 데모만 제공합니다.

이 섹션은 이미지 세트에서 객체 및 장면 감지, 얼굴 분석, 얼굴 비교 같은 Amazon Rekognition's 기능의 하위 집합을 사용하는 방법을 보여 줍니다. 자세한 내용은 Amazon Rekognition 작동 방식 단원을 참조하십시오. Amazon Rekognition API 또는를 사용하여 객체와 장면 AWS CLI 을 감지하고, 얼굴을 감지하고, 얼굴을 비교 및 검색할 수도 있습니다. 자세한 내용은 3단계: AWS CLI 및 AWS SDK API 사용 시작하기 단원을 참조하십시오.

이 섹션은 또한 Rekognition 콘솔을 사용하여 Rekognition의 Amazon CloudWatch 집계 지표를 보는 방법도 보여 줍니다.

주제

- 콘솔 권한 설정
- 연습 1: 객체 및 장면 감지(콘솔)
- 연습 2: 이미지 내 얼굴 분석(콘솔)
- 연습 3: 이미지 내 얼굴 비교(콘솔)
- 연습 4: 집계 지표 보기(콘솔)



콘솔 권한 설정

Rekognition 콘솔을 사용하려면 콘솔에 액세스하는 역할 또는 계정에 대한 적절한 권한이 있어야 합니다. 일부 작업의 경우 Rekognition은 Amazon S3 버킷을 자동으로 생성하여 작업 중에 처리된 파일을 저장합니다. 훈련 파일을 콘솔 버킷이 아닌 다른 버킷에 저장하려면 추가 권한이 필요합니다.

콘솔 액세스 허용

Rekognition 콘솔을 사용하려면 Amazon S3와 Rekognition 콘솔을 포함하는 다음과 같은 IAM 정책을 사용할 수 있습니다. 권한을 할당하는 방법에 대한 자세한 내용은 권한 할당 항목을 참조하세요.

JSON

```
{
    "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:PutBucketCors",
                "s3:GetBucketCors"
            ],
            "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: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": "*"
        }
    ]
}
```

외부 Amazon S3 버킷에 액세스

새 AWS 리전에서 Rekognition 콘솔을 처음 열면 Rekognition은 프로젝트 파일을 저장하는 데 사용되는 버킷(콘솔 버킷)을 생성합니다. 또는 자체 Amazon S3 버킷(외부 버킷)을 사용하여 이미지 또는 매니페스트 파일을 콘솔에 업로드할 수 있습니다. 외부 버킷을 사용하려면 이전 정책에 다음 정책 블록을 추가하세요.

```
{
            "Sid": "s3ExternalBucketPolicies",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
```

```
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectVersion",
                "s3:GetObjectTagging",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket*"
            ]
}
```

권한 할당

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요.

- AWS IAM Identity Center(구 AWS Single Sign-On)에서 사용자 및 그룹:

  권한 세트를 생성합니다. AWS IAM Identity Center(구 AWS Single Sign-On) 사용 설명서의 권한 세트 생성에 나온 지침을 따르세요.

- ID 제공업체를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기(페더레이션)의 지침을 따릅니다.

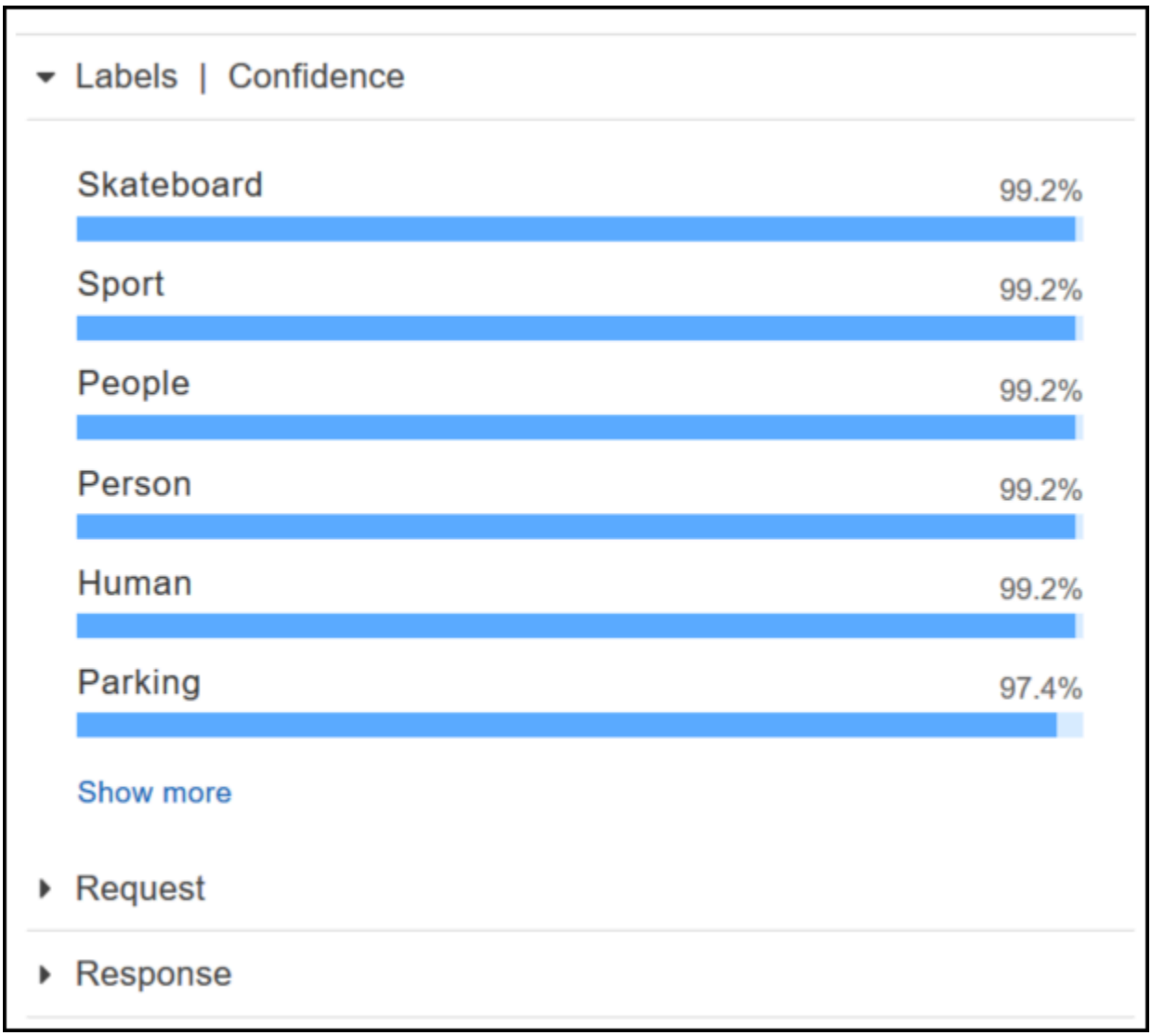
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 IAM 사용자의 역할 생성의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따릅니다.

연습 1: 객체 및 장면 감지(콘솔)

이 섹션은 Amazon Rekognition의 객체 및 장면 감지 기능이 아주 높은 수준에서 어떻게 작동하는지 보여 줍니다. 이미지를 입력으로 지정하면 서비스는 해당 이미지에서 객체와 장면을 감지하고 각 객체와 장면의 백분율 신뢰도 점수와 함께 객체 및 장면을 반환합니다.

예를 들어 Amazon Rekognition은 샘플 이미지에서 객체와 장면(스케이트보드, 스포츠, 사람, 자동차, 차량)을 감지합니다.

Amazon Rekognition은 다음 샘플 응답에 나온 것처럼 샘플 이미지에서 감지된 각 객체의 신뢰도 점수도 반환합니다.



이 응답에 표시된 모든 신뢰도 점수를 보려면 레이블 | 신뢰도 창에서 Show more를 선택합니다.

API에 대한 요청과 API의 응답을 참조로 볼 수도 있습니다.

요청

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

```
         }
      }
   }
}
```

응답


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


자세한 내용은 Amazon Rekognition 작동 방식 단원을 참조하십시오.

제공한 이미지에서 객체 및 장면 감지

Amazon Rekognition 콘솔에 소유하고 있는 이미지를 업로드하거나 이미지의 URL을 입력으로 제공할 수 있습니다. Amazon Rekognition은 제공된 이미지에서 감지한 객체와 장면 및 각 객체와 장면의 신뢰도 점수를 반환합니다.

> **Note**
>
> 이미지는 크기가 5MB 미만이어야 하며 JPEG 또는 PNG 형식이어야 합니다.

제공한 이미지에서 객체 및 장면을 감지하려면

1. https://console.aws.amazon.com/rekognition/에서 Amazon Rekognition 콘솔을 엽니다.
2. 레이블 감지를 선택합니다.

3. 다음 중 하나를 수행하세요.
   - 이미지 업로드 - 업로드를 선택하고 이미지를 저장한 위치로 이동한 다음 이미지를 선택합니다.
   - URL 사용 - 텍스트 상자에 URL을 입력한 다음 이동을 선택합니다.
4. [Labels | Confidence] 창에서 감지된 각 레이블의 신뢰도 점수를 확인합니다.

더 많은 이미지 분석 옵션을 보려면 the section called “이미지 작업” 섹션을 참조하세요.

제공한 비디오에서 사람과 객체 감지

Amazon Rekognition 콘솔에 소유하고 있는 비디오를 업로드하여 입력으로 제공할 수 있습니다. Amazon Rekognition은 비디오에서 감지된 사람, 객체 및 레이블을 반환합니다.

> **Note**
>
> 데모 동영상은 길이가 1분을 초과하거나 용량 30MB를 초과할 수 없습니다. MP4 파일 형식이어야 하며 H.264 코덱을 사용하여 인코딩해야 합니다.

제공한 비디오에서 사람 및 객체를 감지하려면

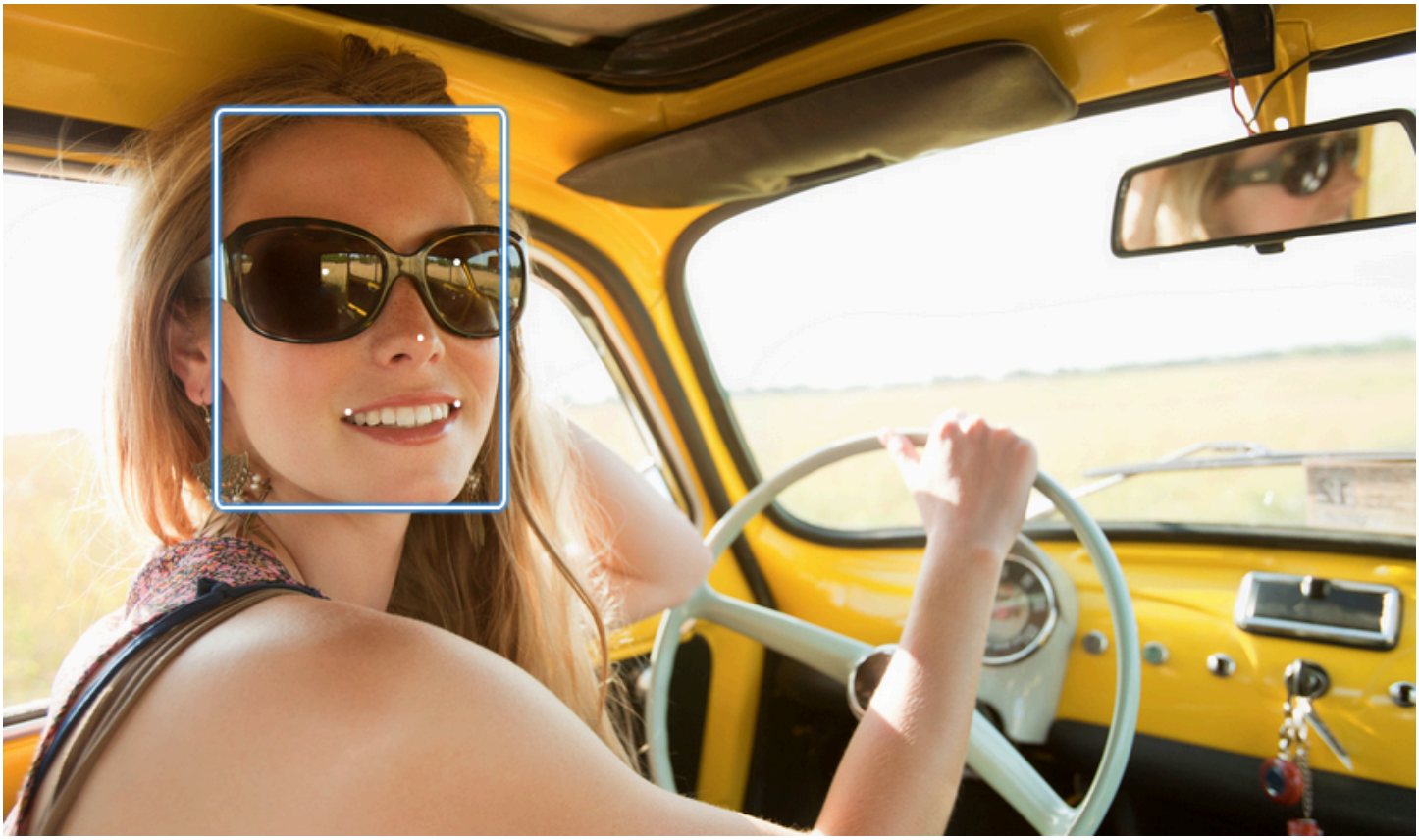
1. https://console.aws.amazon.com/rekognition/에서 Amazon Rekognition 콘솔을 엽니다.
2. 탐색 모음에서 저장된 비디오 분석을 선택합니다.
3. 드롭다운 메뉴의 샘플 선택 또는 자체적으로 업로드에서 자체 비디오를 선택합니다.
4. 비디오를 드래그 앤 드롭하거나 저장한 위치에서 비디오를 선택합니다.

더 많은 비디오 분석 옵션을 보려면 the section called “저장된 비디오 분석 작업”나 the section called “스트리밍 비디오 이벤트 작업” 섹션을 참조하세요.

연습 2: 이미지 내 얼굴 분석(콘솔)

이 섹션에서는 Amazon Rekognition 콘솔을 사용하여 이미지에서 얼굴을 감지하고 얼굴 속성을 분석하는 방법을 보여 줍니다. 얼굴이 포함된 이미지를 입력으로 제공하면 서비스는 이미지에서 얼굴을 감지하고 얼굴의 얼굴 속성을 분석한 다음 이미지에서 감지된 얼굴의 백분율 신뢰도 점수와 얼굴 속성을 반환합니다. 자세한 내용은 Amazon Rekognition 작동 방식 단원을 참조하십시오.

예를 들어 다음 샘플 이미지를 입력으로 선택하면 Amazon Rekognition은 이를 얼굴로 감지하고 해당 얼굴의 신뢰도 점수와 감지된 얼굴 속성을 반환합니다.



다음은 샘플 응답을 보여 줍니다.

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

입력 이미지에 얼굴이 여러 개 있는 경우, Rekognition은 이미지에서 최대 100개의 얼굴을 감지합니다. 감지된 각 얼굴은 사각형으로 표시됩니다. 얼굴에서 사각형으로 표시된 영역을 클릭하면 Rekognition은 얼굴 | 신뢰도창에 해당 얼굴의 신뢰도 점수와 감지된 얼굴 속성을 표시합니다.

제공한 이미지에서 얼굴 분석

Amazon Rekognition 콘솔에 자체 이미지를 업로드하거나 이미지 URL을 제공할 수 있습니다.

> **Note**
>
> 이미지는 크기가 5MB 미만이어야 하며 JPEG 또는 PNG 형식이어야 합니다.

제공한 이미지의 얼굴을 분석하려면

1. https://console.aws.amazon.com/rekognition/에서 Amazon Rekognition 콘솔을 엽니다.
2. [Facial analysis]를 선택합니다.
3. 다음 중 하나를 수행하세요.
   - 이미지 업로드 - 업로드를 선택하고 이미지를 저장한 위치로 이동한 다음 이미지를 선택합니다.
   - URL 사용 - 텍스트 상자에 URL을 입력한 다음 이동을 선택합니다.
4. [Faces | Confidence] 창에서 감지된 얼굴 중 하나의 신뢰도 점수와 얼굴 속성을 확인합니다.
5. 이미지에 여러 얼굴이 있는 경우, 다른 얼굴 중 하나를 선택해 속성과 점수를 봅니다.

연습 3: 이미지 내 얼굴 비교(콘솔)

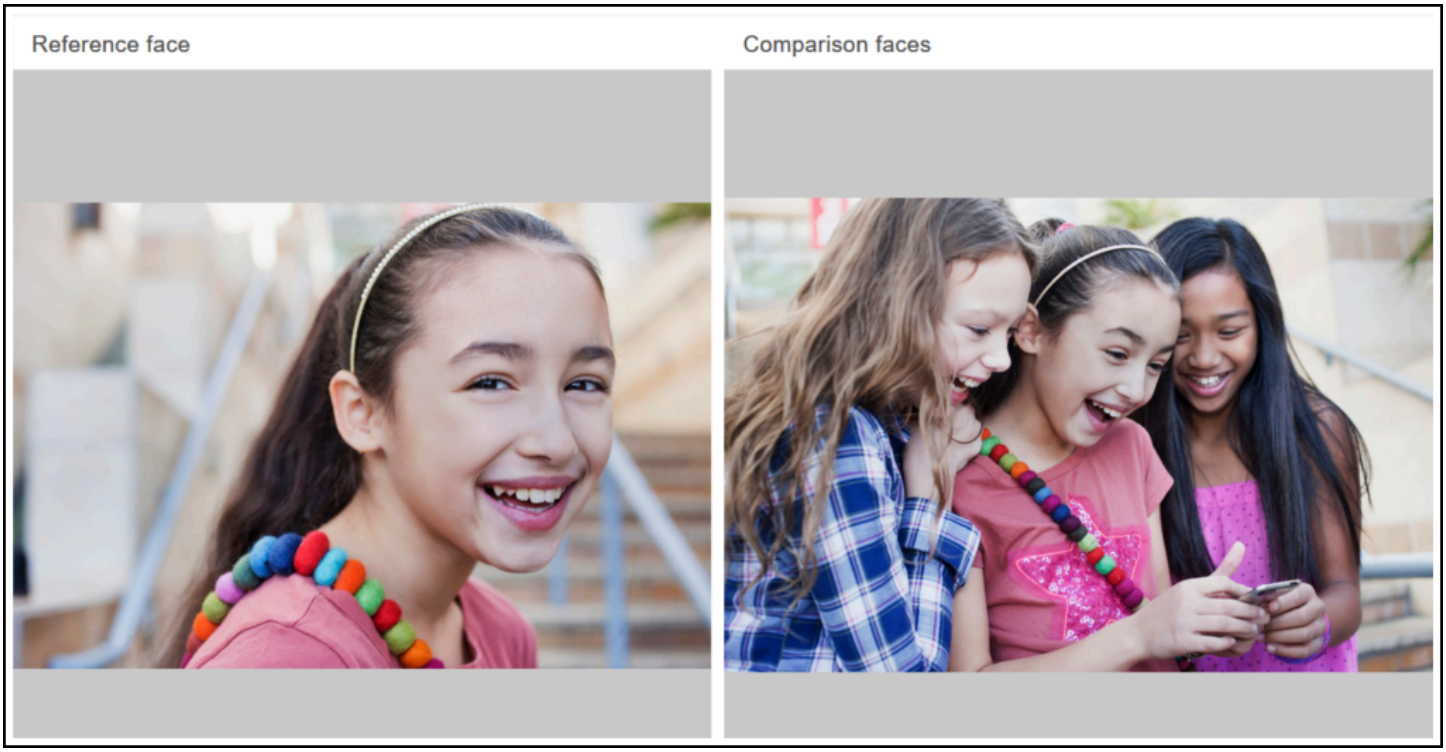
이 섹션에서는 Amazon Rekognition 콘솔을 사용하여 여러 얼굴이 포함된 이미지 집합 내에서 얼굴을 비교하는 방법을 보여 줍니다. 참조 얼굴(소스)과 비교 얼굴(대상) 이미지를 지정하면 Rekognition은 소스 이미지에서 가장 큰 얼굴(즉, 참조 얼굴)을 대상 이미지에서 감지된 최대 100개 얼굴(즉, 비교 얼굴)과 비교한 다음 소스의 얼굴이 대상 이미지의 얼굴과 얼마나 비슷한지 찾습니다. 각 비교의 유사성 점수는 [Results] 창에 표시됩니다.

대상 이미지에 여러 얼굴이 있는 경우, Rekognition은 소스 이미지의 얼굴을 대상 이미지의 최대 100개 얼굴과 비교한 다음 각 일치에 유사성 점수를 할당합니다.

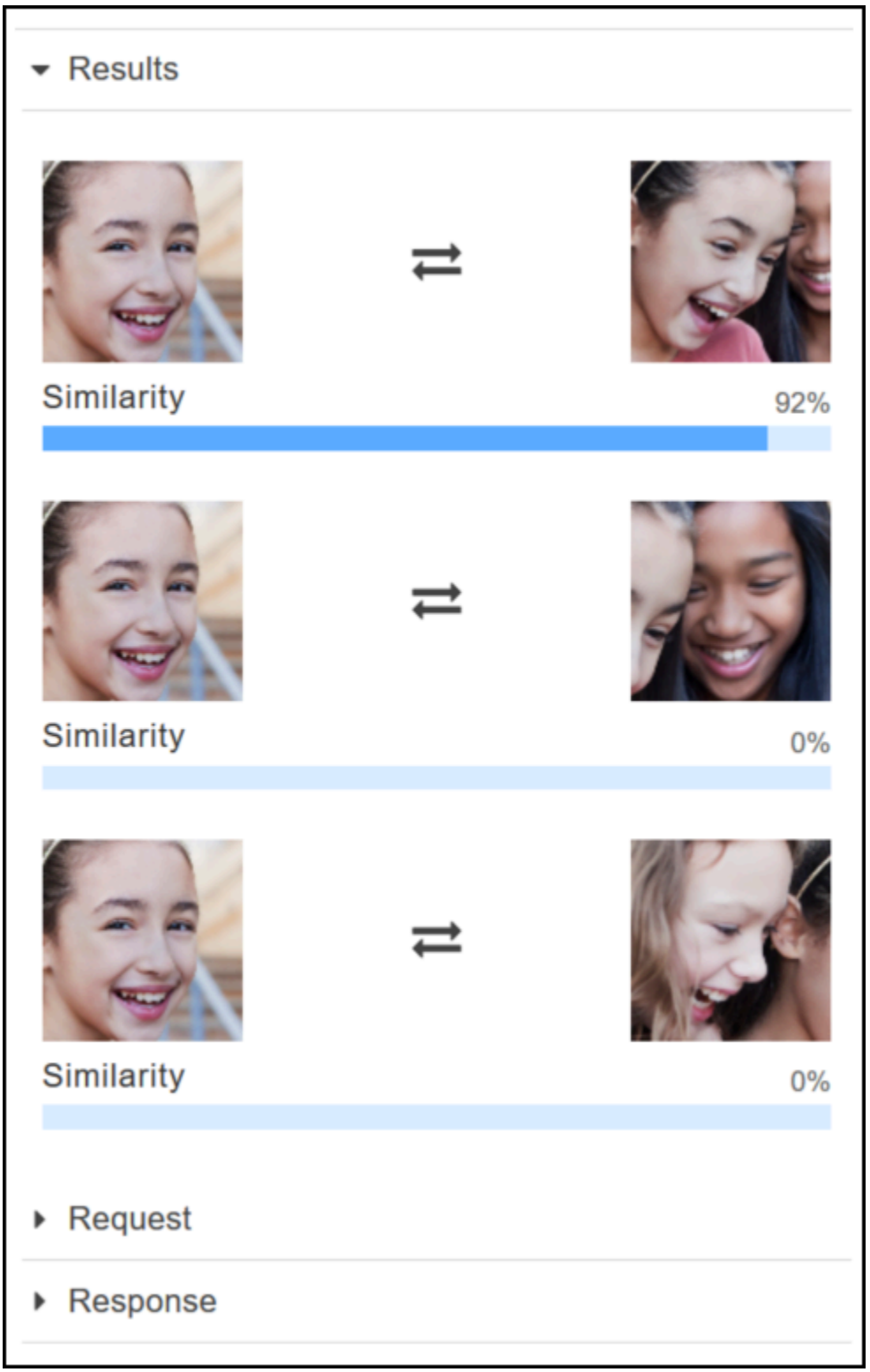
원본 이미지에 여러 얼굴이 포함되어 있는 경우, 서비스는 원본 이미지에서 가장 큰 얼굴을 감지하여 이를 사용해 대상 이미지에서 감지된 각각의 얼굴과 비교합니다.

자세한 내용은 이미지에 있는 얼굴 비교 단원을 참조하십시오.

예를 들어 왼쪽에 소스 이미지로 표시된 샘플 이미지와 오른쪽에 대상 이미지로 표시된 샘플 이미지의 경우, Rekognition은 소스 이미지의 얼굴을 감지하여 대상 이미지에서 감지된 각각의 얼굴과 비교한 다음 각 쌍의 유사성 점수를 표시합니다.



다음은 대상 이미지에서 감지된 얼굴들과 각 얼굴의 유사성 점수를 보여 줍니다.



제공한 이미지 내 얼굴 비교

자체 소스 이미지와 대상 이미지를 업로드하면 Rekognition이 이미지에서 얼굴을 비교할 수 있습니다. 또는 이미지 위치의 URL을 지정할 수도 있습니다.

> **Note**
>
> 이미지는 크기가 5MB 미만이어야 하며 JPEG 또는 PNG 형식이어야 합니다.

이미지에서 얼굴을 비교하려면

1. https://console.aws.amazon.com/rekognition/에서 Amazon Rekognition 콘솔을 엽니다.
2. [Face comparison]을 선택합니다.
3. 원본 이미지에 대해 다음 중 하나를 수행합니다.
   - 이미지 업로드 - 왼쪽에서 업로드를 선택하고 소스 이미지를 저장한 위치로 이동한 다음 이미지를 선택합니다.
   - URL 사용 - 텍스트 상자에 소스 이미지의 URL을 입력한 다음 이동을 선택합니다.
4. 대상 이미지에 대해 다음 중 하나를 수행합니다.
   - 이미지 업로드 - 오른쪽에서 업로드를 선택하고 소스 이미지를 저장한 위치로 이동한 다음 이미지를 선택합니다.
   - URL 사용 - 텍스트 상자에 소스 이미지의 URL을 입력한 다음 이동을 선택합니다.
5. Rekognition은 소스 이미지에서 가장 큰 얼굴을 대상 이미지의 최대 100개 얼굴과 비교한 다음 결과 창에 각 쌍의 유사성 점수를 표시합니다.

연습 4: 집계 지표 보기(콘솔)

Amazon Rekognition 지표 창은 지정된 기간 동안의 개별 Rekognition 지표 집계를 위한 활동 그래프를 표시합니다. 예를 들어 `SuccessfulRequestCount` 집계 지표는 지난 7일 동안 모든 Rekognition API 작업에 대한 성공적 요청 총수를 보여 줍니다.

다음 표에는 Rekognition 지표 창에 표시되는 그래프와 그에 해당하는 Rekognition 지표가 나열되어 있습니다. 자세한 내용은 Rekognition의 CloudWatch 지표 단원을 참조하십시오.

그래프	집계된 측정치
성공적 호출	SuccessfulRequestCount
클라이언트 오류	UserErrorCount

그래프	집계된 측정치
서버 오류	ServerErrorCount
병목 현상 발생	ThrottledCount
감지된 레이블	DetectedLabelCount
감지된 얼굴	DetectedFaceCount

각 그래프는 지정된 기간 동안 수집된 집계 측정치 데이터를 보여 줍니다. 해당 기간 동안 집계된 측정치 데이터의 총 개수도 표시됩니다. 개별 API 호출의 측정치를 보려면 각 그래프 아래의 링크를 선택합니다.

사용자가 Rekognition 지표 창에 액세스하도록 허용하려면 해당 사용자에게 적절한 CloudWatch 및 Rekognition 권한이 있어야 합니다. 예를 들어 `AmazonRekognitionReadOnlyAccess` 및 `CloudWatchReadOnlyAccess` 관리형 정책 권한이 있는 사용자는 측정치 창을 볼 수 있습니다. 필요한 권한이 없는 사용자는 측정치 창을 열어도 그래프가 나타나지 않습니다. 자세한 내용은 Amazon Rekognition의 자격 증명 및 액세스 관리 단원을 참조하십시오.

CloudWatch를 통한 Rekognition 모니터링에 대한 자세한 내용은 Amazon CloudWatch를 사용한 Rekognition 모니터링 섹션을 참조하세요.

집계 측정치를 보려면(콘솔)

1. https://console.aws.amazon.com/rekognition/에서 Amazon Rekognition 콘솔을 엽니다.
2. 탐색 창에서 지표(Metrics)를 선택합니다.
3. 드롭다운 목록에서 원하는 측정치 기간을 선택합니다.
4. 그래프를 업데이트하려면 [Refresh] 버튼을 선택합니다.
5. 특정 집계 지표의 상세한 CloudWatch 지표를 보려면 지표 그래프 아래의 CloudWatch에서 세부 정보 확인을 선택합니다.

이미지 및 비디오 작업

사용자는 이미지, 저장된 비디오, 스트리밍 비디오라는 세 가지 미디어 유형에 Amazon Rekognition API 작업을 사용할 수 있습니다. 이 섹션에서는 Amazon Rekognition에 액세스하여 다양한 유형의 미디어를 처리하는 코드 작성에 대한 일반적인 정보를 제공합니다. 모범 사례 및 고려 사항에 대한 지침은 처리 중인 미디어 유형에 따라 아래 나열된 각 섹션을 참조하세요.

이 안내서의 다른 단원에서는 얼굴 감지처럼 특정한 유형의 이미지 및 비디오 분석에 대한 정보를 제공합니다.

주제

- 이미지 작업
- 저장된 비디오 분석 작업
- 스트리밍 비디오 이벤트 작업
- 오류 처리
- FedRAMP 인증 서비스로 Amazon Rekognition 사용

이미지 작업

이 섹션에서는 Amazon Rekognition Image가 이미지에 대해 수행할 수 있는 분석 유형을 다룹니다.

- 객체 및 장면 감지
- 얼굴 감지 및 비교
- 컬렉션에서 얼굴 검색
- 유명 인사 인식
- 이미지 조절
- 이미지 속 텍스트 감지

이러한 작업은 Amazon Rekognition Image가 작업에서 발견된 어떠한 정보도 유지하지 않는 비스토리지 API 작업에 의해 수행됩니다. 입력 이미지 바이트는 비 스토리지 API 작업에 의해 유지되지 않습니다. 자세한 내용은 비스토리지 및 스토리지 API 작업 이해 단원을 참조하십시오.

Amazon Rekognition Image는 또한 나중에 검색할 수 있도록 얼굴 메타데이터를 컬렉션에 저장할 수 있습니다. 자세한 내용은 컬렉션에서 얼굴 검색 단원을 참조하십시오.

이 섹션에서는 Amazon Rekognition Image API 작업을 사용하여 Amazon S3 버킷에 저장된 이미지와 로컬 파일 시스템에서 로드된 이미지 바이트를 분석합니다. 이 섹션에서는 .jpg 이미지에서 이미지 방향 정보를 가져오는 방법에 대해서도 알아봅니다.

Rekognition은 RGB 채널만 사용하여 추론을 수행합니다. AWS 는 사용자가 디스플레이를 사용하여 비교를 시각적으로(사람이 수동으로) 검사하기 전에 알파 채널을 제거하도록 권장합니다.

주제

- 이미지 사양
- Amazon S3 버킷에 저장된 이미지 분석
- 로컬 파일 시스템에서 불러온 이미지 분석
- 경계 상자 표시
- 이미지 방향 및 경계 상자 좌표 가져오기

이미지 사양

Amazon Rekognition Image 작업은 .jpg나 .png 형식의 입력 이미지를 분석할 수 있습니다.

직접 호출의 일부로 Amazon Rekognition Image 작업에 이미지 바이트를 전달하거나 기존 Amazon S3 객체를 참조합니다. Amazon S3 버킷에 저장된 이미지를 분석하는 예제는 Amazon S3 버킷에 저장된 이미지 분석 섹션을 참조하세요. 이미지 바이트를 Amazon Rekognition Image API 작업으로 전달하는 예제는 로컬 파일 시스템에서 불러온 이미지 분석 섹션을 참조하세요.

HTTP를 사용하여 Amazon Rekognition Image 작업의 일부로 이미지 바이트를 전달하는 경우 이미지 바이트는 base64로 인코딩된 문자열이어야 합니다. AWS SDK를 사용하여 API 작업 호출의 일부로 이미지 바이트를 전달하는 경우 이미지 바이트를 base64로 인코딩해야 하는지 여부는 사용하는 언어에 따라 달라집니다.

다음 AWS SDKs는 자동으로 이미지를 base64 인코딩하므로 Amazon Rekognition Image API 작업을 호출하기 전에 이미지 바이트를 인코딩할 필요가 없습니다.

- Java
- JavaScript
- Python
- PHP

다른 AWS SDK를 사용하는 경우 Rekognition API 작업을 직접 호출할 때 이미지 형식 오류가 발생하면 이미지 바이트를 Rekognition API 작업에 전달하기 전에 base64로 인코딩하세요.

AWS CLI 를 사용하여 Amazon Rekognition Image 작업을 호출하는 경우 호출의 일부로 이미지 바이트를 전달하는 것은 지원되지 않습니다. 먼저 이미지를 Amazon S3 버킷에 업로드한 다음 업로드된 이미지를 참조하는 작업을 호출해야 합니다.

> **Note**
>
> 이미지 바이트 대신 `S3Object`에 저장된 이미지를 전달하면 이미지를 base64로 인코딩할 필요가 없습니다.

Amazon Rekognition Image 작업의 지연 시간을 최소화하는 방법에 대한 자세한 내용은 Amazon Rekognition Image 작업 지연 시간 섹션을 참조하세요.

이미지 방향 수정

여러 Rekognition API 작업에서는 분석된 이미지의 방향이 반환됩니다. 이미지 방향을 알면 이미지 방향을 전환하여 표시할 수 있으므로 중요합니다. 얼굴을 분석하는 Rekognition API 작업은 이미지 속의 얼굴 위치에 대한 경계 상자도 반환합니다. 경계 상자를 사용하여 이미지의 얼굴 주위에 상자를 표시할 수 있습니다. 반환된 경계 상자 좌표는 이미지 방향의 영향을 받으며 얼굴 주위에 상자를 올바르게 표시하려면 경계 상자 좌표를 변환해야 할 수도 있습니다. 자세한 내용은 이미지 방향 및 경계 상자 좌표 가져오기 단원을 참조하십시오.

이미지 크기 조정

분석 중에 Amazon Rekognition은 특정 모델 또는 알고리즘에 가장 적합한 사전 정의된 범위 세트를 사용하여 내부적으로 이미지 크기를 조정합니다. 이로 인해 Amazon Rekognition은 입력 이미지의 해상도에 따라 다른 수의 객체를 감지하거나 다른 결과를 내놓을 수 있습니다. 예를 들어 이미지가 두 개 있다고 가정하겠습니다. 첫 번째 이미지의 해상도는 1024x768픽셀입니다. 첫 번째 이미지의 크기가 조정된 버전인 두 번째 이미지의 해상도는 640x480픽셀입니다. DetectLabels에 이미지를 제출하는 경우 `DetectLabels`에 보낸 두 직접 호출의 응답이 약간 다를 수 있습니다.

Amazon S3 버킷에 저장된 이미지 분석

Amazon Rekognition Image는 Amazon S3 버킷에 저장된 이미지 또는 이미지 바이트로 제공된 이미지를 분석할 수 있습니다.

이 주제에서는 DetectLabels API 작업을 사용하여 Amazon S3 버킷에 저장된 이미지(JPEG 또는 PNG)에서 객체, 개념, 장면을 감지합니다. Image 입력 파라미터를 사용하여 Amazon Rekognition Image API 작업에 이미지를 전달합니다. Image 내에서 S3Object 객체 속성을 지정하여 S3 버킷에 저장된 이미지를 참조합니다. Amazon S3 버킷에 저장된 이미지의 이미지 바이트는 base64로 인코딩할 필요가 없습니다. 자세한 내용은 이미지 사양 단원을 참조하십시오.

요청 예제

DetectLabels에 대한 이 예제 JSON 요청에서는 amzn-s3-demo-bucket이라는 Amazon S3 버킷에서 소스 이미지(input.jpg)를 불러옵니다. 단, S3 객체가 있는 S3 버킷의 리전과 Amazon Rekognition Image 작업에 사용하는 리전이 일치해야 한다는 것을 알아 두세요.

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

다음 예제에서는 AWS SDKs 및를 사용하여 AWS CLI 를 호출합니다DetectLabels. DetectLabels 작업 응답에 대한 자세한 내용은 DetectLabels 응답 단원을 참조하십시오.

이미지에서 레이블을 감지하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오. API 작업을 직접 호출하는 사용자에게 프로그래밍 방식 액세스에 대한 적절한 권한을 부여했는지 확인하세요. 이를 수행하는 방법에 대한 지침은 프로그래밍 방식 액세스 권한 부여 섹션을 참조하세요.

2. 나무, 집, 보트 등과 같은 객체가 한 개 이상 있는 이미지를 S3 버킷에 업로드합니다. 이미지는 .jpg 또는 .png 형식이어야 합니다.

이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 예제를 사용하여 DetectLabels 작업을 호출합니다.

Java

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 `detect-labels` CLI 작업의 JSON 출력을 표시합니다. `bucket` 및 `photo`의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
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
```

Windows를 사용하는 경우 아래 예와 같이 따옴표를 이스케이프 처리해 주어야 할 수 있습니다.

```
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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//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

이 예제는 입력 이미지에서 감지된 레이블을 표시합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

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

import boto3

def detect_labels(photo, bucket):

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

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

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

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

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

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

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

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

        return len(response['Labels'])

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

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

Node.Js

이 예제는 이미지에서 감지된 레이블에 대한 정보를 표시합니다.

photo의 값을, 하나 이상의 유명 인사 얼굴을 포함하는 이미지 파일의 경로와 파일 이름으로 변경합니다. bucket의 값을 제공된 이미지 파일이 저장된 S3 버킷의 이름으로 바꿉니다.

REGION의 값을 귀하의 계정과 연결된 리전 이름으로 변경하세요. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
// 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

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다.

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

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

public class DetectLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
        {
            Image = new Image()
            {
```

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

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

Ruby

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다.

```
  # 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
```

응답의 예

DetectLabels의 응답은 이미지에서 감지된 레이블의 배열과 각각의 해당 신뢰도 수준입니다.

이미지에서 DetectLabels 작업을 수행하면 Amazon Rekognition은 다음 예제 응답과 유사한 출력을 반환합니다.

이 응답은 작업에서 여러 개의 레이블(인물, 차량, 자동차)을 감지했음을 보여 줍니다. 각 레이블에는 연결된 신뢰도 수준이 있습니다. 예를 들어 감지 알고리즘 상 이미지에 사람이 포함될 신뢰도는 98.991432%입니다.

또한 응답에는 Parents 배열에 레이블의 상위 레이블이 포함되어 있습니다. 예를 들어 자동차 레이블은 차량 및 운송이라는 두 개의 상위 레이블을 갖습니다.

일반적 객체 레이블에 대한 응답은 입력 이미지 상의 레이블 위치에 대한 경계 상자 정보를 포함합니다. 예를 들어 인물 레이블은 두 개의 경계 상자를 포함하는 인스턴스 배열을 갖습니다. 이들은 이미지에서 감지된 두 사람의 위치입니다.

LabelModelVersion 필드에는 DetectLabels가 사용하는 감지 모델의 버전 번호가 포함됩니다.

DetectLabels 작업의 사용에 대한 자세한 내용은 객체 및 개념 감지 섹션을 참조하세요.


```
{

    {
    "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"
}
```

```
}
```

로컬 파일 시스템에서 불러온 이미지 분석

Amazon Rekognition Image 작업은 Amazon S3 버킷에 저장된 이미지 또는 이미지 바이트로 제공된 이미지를 분석할 수 있습니다.

이 주제에서는 로컬 파일 시스템에서 로드된 파일을 사용하여 이미지 바이트를 Amazon Rekognition Image API 작업에 제공하는 예제를 설명합니다. Image 입력 파라미터를 사용하여 Amazon Rekognition Image API 작업에 이미지 바이트를 전달합니다. Image에서 Bytes 속성을 지정하여 base64로 인코딩된 이미지 바이트를 전달합니다.

Bytes 입력 파라미터를 사용하여 Amazon Rekognition API 작업에 전달하는 이미지 바이트는 base64로 인코딩해야 합니다. 이러한 예제의 AWS SDK는 자동으로 base64 인코딩 이미지를 사용합니다. Amazon Rekognition API 작업을 직접 호출하기 전에 이미지 바이트를 인코딩할 필요가 없습니다. 자세한 내용은 이미지 사양 단원을 참조하십시오.

DetectLabels에 대한 이 예제 JSON 요청에서 소스 이미지 바이트는 Bytes 입력 파라미터로 전달됩니다.

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

다음 예제에서는 AWS SDKs 및를 사용하여 AWS CLI 를 호출합니다DetectLabels. DetectLabels 작업 응답에 대한 자세한 내용은 DetectLabels 응답 단원을 참조하십시오.

클라이언트 JavaScript 예제는 JavaScript 사용 단원을 참조하십시오.

로컬 이미지의 레이블을 감지하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

    a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 DetectLabels 작업을 호출합니다.

Java

다음 Java 예제는 로컬 파일 시스템에서 이미지를 로드하고 detectLabels AWS SDK 작업을 사용하여 레이블을 감지하는 방법을 보여줍니다. photo의 값을 이미지 파일(.jpg 또는 .png 형식)의 경로와 파일 이름으로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

다음 Python용 AWS SDK 예제는 로컬 파일 시스템에서 이미지를 로드하고 detect_labels 작업을 호출하는 방법을 보여줍니다. photo의 값을 이미지 파일(.jpg 또는 .png 형식)의 경로와 파일 이름으로 바꿉니다.

```
#Copyright 2018 Amazon.com, Inc. or its 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

다음 예제에서는 로컬 파일 시스템에서 이미지를 로드하고 DetectLabels 작업을 사용하여 레이블을 감지하는 방법을 보여줍니다. photo의 값을 이미지 파일(.jpg 또는 .png 형식)의 경로와 파일 이름으로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

다음 PHP용 AWS SDK 예제는 로컬 파일 시스템에서 이미지를 로드하고 DetectFaces API 작업을 호출하는 방법을 보여줍니다. photo의 값을 이미지 파일(.jpg 또는 .png 형식)의 경로와 파일 이름으로 바꿉니다.

```
<?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

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. photo의 값을 이미지 파일(.jpg 또는 .png 형식)의 경로와 파일 이름으로 바꿉니다.

```
#Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

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

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

```
 */
public class DetectLabels {
    public static void main(String[] args) {
        final String usage = """
            Usage: <bucketName> <sourceImage>

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

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

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

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

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

```
                    .name(sourceImage)
                    .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

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

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

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

JavaScript 사용

다음 JavaScript 웹 페이지 예제를 이용하여 이미지를 선택하고, 이미지에서 감지된 얼굴의 예상 연령을 확인할 수 있습니다. DetectFaces를 직접 호출하면 추정 연령이 반환됩니다.

선택한 이미지는, 이미지를 base64로 인코딩하는 JavaScript `FileReader.readAsDataURL` 함수를 사용하여 로드됩니다. HTML 캔버스에 이미지를 표시할 때 유용한 기능입니다. 그러나 이렇게 하면 Amazon Rekognition Image 작업으로 전달하기 전에 이미지 바이트 인코딩을 해제해야 합니다. 이 예제에서는 로드된 이미지 바이트를 해독하는 방법을 보여 줍니다. 인코딩된 이미지 바이트가 불편하면 그 대신 로딩된 이미지를 인코딩하지 않는 `FileReader.readAsArrayBuffer`를 사용하십시오. 이렇게 하면 이미지 바이트 인코딩을 먼저 해제하지 않고도 Amazon Rekognition Image 작업을 직접 호출할 수 있습니다. 예제는 readAsArrayBuffer 사용 섹션을 참조하세요.

JavaScript 예제를 실행하려면

1. 예제 소스 코드를 편집기에 로드합니다.
2. Amazon Cognito 자격 증명 풀 식별자를 가져옵니다. 자세한 내용은 Amazon Cognito 자격 증명 풀 식별자 가져오기 단원을 참조하십시오.
3. 예제 코드의 AnonLog 함수에서 `IdentityPoolIdToUse`와 `RegionToUse`를 Amazon Cognito 자격 증명 풀 식별자 가져오기의 9단계에서 기록해 둔 값으로 바꿉니다.
4. `DetectFaces` 함수에서 `RegionToUse`를 이전 단계에서 사용한 값으로 바꿉니다.
5. 예제 소스 코드를 `.html` 파일로 저장합니다.
6. 그 파일을 브라우저에 로드합니다.
7. 찾아보기... 버튼을 선택하고 얼굴이 하나 이상 들어 있는 이미지를 선택합니다. 이미지에서 감지된 각 얼굴의 예상 연령을 담은 표가 나타납니다.

> **Note**
>
> 다음 코드 예제는 더 이상 Amazon Cognito의 일부가 아닌 두 스크립트를 사용합니다. 이러한 파일을 가져오려면 aws-cognito-sdk.min.js 및 amazon-cognito-identity.min.js의 링크를 따라 이동한 후, 각각의 텍스트를 별도의 `.js` 파일로 저장합니다.

JavaScript 예제 코드

다음 코드 예제는 JavaScript V2를 사용합니다. JavaScript V3의 예제는 AWS 설명서 SDK 예제 GitHub 리포지토리의 예제를 참조하세요.

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

```
</head>

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

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

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

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

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

  // Configure the credentials provider to use your identity pool
  AWS.config.region = 'RegionToUse'; // Region
  AWS.config.credentials = new AWS.CognitoIdentityCredentials({
    IdentityPoolId: 'IdentityPoolIdToUse',
  });
```

```
    // Make the call to obtain credentials
    AWS.config.credentials.get(function () {
      // Credentials will be available when this function is called.
      var accessKeyId = AWS.config.credentials.accessKeyId;
      var secretAccessKey = AWS.config.credentials.secretAccessKey;
      var sessionToken = AWS.config.credentials.sessionToken;
    });
  }
</script>
</html>
```

readAsArrayBuffer 사용

다음은 JavaScript V2를 사용하여 샘플 코드에서 `ProcessImage` 함수를 대안적으로 구현하는 코드 조각입니다. 여기서는 `readAsArrayBuffer`를 사용하여 이미지를 로드하고 `DetectFaces`를 호출합니다. `readAsArrayBuffer`는 로드된 파일을 base64로 인코딩하지 않기 때문에 Amazon Rekognition Image 작업을 직접 호출하기 위해 먼저 이미지 바이트의 인코딩을 해제할 필요가 없습니다.

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

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

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

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

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

Amazon Cognito 자격 증명 풀 식별자 가져오기

간단히 설명하기 위해, 이 예제에서는 익명 Amazon Cognito 자격 증명 풀을 사용하여 Amazon Rekognition Image API에 대한 인증되지 않은 액세스 권한을 제공합니다. 이 방법이 현재 상황에 알맞습니다. 예를 들어, 사용자 로그인을 위한 웹 사이트 무료 액세스 권한 또는 평가판 액세스 권한을 미인증 액세스 권한으로 제공할 수 있습니다. 인증되지 않은 액세스 권한을 제공하려면 Amazon Cognito 사용자 풀을 사용하세요. 자세한 내용은 Amazon Cognito 사용자 풀을 참조하세요.

다음 절차에서는 인증되지 않은 자격 증명을 사용할 수 있는 자격 증명 풀을 만드는 방법과 예제 코드에 필요한 자격 증명 풀 식별자를 가져오는 방법을 보여 줍니다.

자격 증명 풀 식별자를 가져오려면

1. Amazon Cognito 콘솔을 엽니다.
2. 새 자격 증명 풀 생성을 선택합니다.
3. 자격 증명 풀 이름*에 자격 증명 풀의 이름을 입력합니다.
4. 인증되지 않은 자격 증명에서 인증되지 않은 자격 증명에 대한 액세스 활성화를 선택합니다.
5. 풀 생성을 선택합니다.
6. 세부 정보 보기를 선택하고 인증되지 않은 자격 증명의 역할 이름을 적어 둡니다.
7. 허용을 선택합니다.

8. 플랫폼에서 JavaScript를 선택합니다.
9. AWS 자격 증명 얻기에서 코드 조각에 표시된 `AWS.config.region` 및 `IdentityPooldId`의 값을 적어 둡니다.
10. IAM 콘솔(https://console.aws.amazon.com/iam/)을 엽니다.
11. 탐색 창에서 역할을 선택합니다.
12. 6단계에서 적어 둔 역할 이름을 선택합니다.
13. 권한 탭에서 정책 연결을 선택합니다.
14. AmazonRekognitionReadOnlyAccess를 선택합니다.
15. 정책 연결을 선택합니다.

경계 상자 표시

Amazon Rekognition Image 작업은 이미지에서 감지된 항목에 대해 경계 상자 좌표를 반환할 수 있습니다. 예를 들어 DetectFaces 작업은 이미지에서 감지된 각 얼굴에 대해 경계 상자(BoundingBox)를 반환합니다. 경계 상자 좌표를 사용하여 감지된 항목 주위에 상자를 표시할 수 있습니다. 예를 들어 다음 이미지에는 얼굴을 둘러싼 경계 상자가 표시되어 있습니다.

BoundingBox에는 다음과 같은 속성이 있습니다.

- 높이 - 전체 이미지 높이에 대한 비율로서 경계 상자의 높이입니다.
- 좌측 - 전체 이미지 너비에 대한 비율로서 경계 상자의 좌측 좌표입니다.
- 상단 - 전체 이미지 높이에 대한 비율로서 경계 상자의 상단 좌표입니다.
- 너비 - 전체 이미지 너비에 대한 비율로서 경계 상자의 너비입니다.

각 BoundingBox 속성은 0~1 사이의 값을 가집니다. 각 속성 값은 전체 이미지 너비(Left 및 Width) 또는 높이(Height 및 Top)에 대한 비율입니다. 예를 들어 입력 이미지가 700x200픽셀이고, 경계 상자의 상단-좌측 좌표가 350x50픽셀일 경우 API는 Left 값 0.5(350/700)와 Top 값 0.25(50/200)를 반환합니다.

다음 다이어그램에는 각 경계 상자가 커버하는 이미지 범위가 나와 있습니다.

경계 상자를 올바른 위치 및 크기로 표시하려면, (원하는 값에 따라) BoundingBox 값을 이미지 너비 또는 높이에 곱해 픽셀 값을 구해야 합니다. 이 픽셀 값을 사용하여 경계 상자를 표시합니다. 예를 들어 이전 이미지의 픽셀 치수는 608(너비) x 588(높이)입니다. 얼굴의 경계 상자 값은 다음과 같습니다.

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

얼굴 경계 상자의 위치는 다음과 같이 계산됩니다.

```
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
```

이들 값을 사용하여 얼굴 주위에 경계 상자를 표시합니다.

> **Note**
>
> 이미지 방향을 여러 방법으로 설정할 수 있습니다. 애플리케이션이 이미지를 올바르게 표시하기 위해 이미지를 회전해야 할 수 있습니다. 경계 상자 좌표는 이미지의 방향에 영향을 받습니다. 경계 상자를 올바른 위치에 표시하기 위해 좌표를 변환해야 할 수 있습니다. 자세한 내용은 이미지 방향 및 경계 상자 좌표 가져오기 단원을 참조하십시오.

다음 예제는 DetectFaces를 직접 호출하여 감지된 얼굴 주위에 경계 상자를 표시하는 방법을 보여줍니다. 예제에서는 이미지 방향이 0도로 설정된 것으로 가정합니다. 또한 Amazon S3 버킷에서 이미지를 다운로드하는 방법도 보여줍니다.

경계 상자를 표시하는 방법

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 DetectFaces 작업을 호출합니다.

   Java

   bucket의 값을 이미지 파일이 저장된 Amazon S3 버킷으로 바꿉니다. photo의 값을 이미지 파일(.jpg 또는 .png 형식)의 파일 이름으로 바꿉니다.

```
//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

bucket의 값을 이미지 파일이 저장된 Amazon S3 버킷으로 바꿉니다. photo의 값을 이미지 파일(.jpg 또는 .png 형식)의 파일 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

참고로, s3은 AWS SDK Amazon S3 클라이언트를 가리키며 rekClient는 AWS SDK Amazon Rekognition 클라이언트를 가리킵니다.

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

/**
```

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

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

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

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

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

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

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

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

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

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

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

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

            result = rekClient.detectFaces(facesRequest);

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

                System.out.println("Left: " + (int) left);
```

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

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

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

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

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

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

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

```
            return null;
        }

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

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

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

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

이미지 방향 및 경계 상자 좌표 가져오기

Amazon Rekognition Image를 사용하는 애플리케이션은 일반적으로 Amazon Rekognition Image 작업으로 감지된 이미지 및 감지된 얼굴 주위의 상자를 표시해야 합니다. 애플리케이션에서 이미지를 올바르게 표시하려면 이미지의 방향을 알아야 합니다. 이 방향을 수정해야 할 수도 있습니다. 일부 .jpg 파일의 경우 이미지의 교환 이미지 파일 형식(Exif) 메타데이터에 이미지의 방향이 포함됩니다.

얼굴 주위에 상자를 표시하려면 얼굴의 경계 상자에 대한 좌표가 필요합니다. 상자 방향이 올바르지 않으면 해당 좌표를 조정해야 할 수 있습니다. Amazon Rekognition Image의 얼굴 감지 작업은 감지된 각 얼굴에 대한 경계 상자 좌표를 반환하지만 Exif 메타데이터가 없는.jpg 파일의 좌표는 추정하지 않습니다.

다음 예제는 이미지에서 감지된 얼굴의 경계 상자 좌표를 가져오는 방법을 보여줍니다.

이 예제의 정보를 사용하여 이미지의 방향이 올바른지, 경계 상자가 애플리케이션의 올바른 위치에 표시되는지 확인하십시오.

이미지와 경계 상자를 회전하고 표시하는 데 사용되는 코드는 사용하는 언어와 환경에 따라 다르므로 코드에 이미지와 경계 상자를 표시하는 방법이나 Exif 메타데이터에서 방향 정보를 가져오는 방법을 설명하지 않습니다.

이미지의 방향 찾기

애플리케이션에서 이미지를 올바르게 표시하려면 이미지를 회전해야 할 수 있습니다. 다음 이미지의 방향은 0도이며 올바르게 표시되어 있습니다.

그러나 다음 이미지는 시계 반대 방향으로 90도 회전된 상태입니다. 올바르게 표시하려면 이미지의 방향을 찾고 코드에서 해당 정보를 사용하여 이미지를 0도로 회전해야 합니다.

.jpg 형식의 일부 이미지는 Exif 메타데이터에 방향 정보를 포함합니다. 가능한 경우 이미지의 Exif 메타데이터에는 방향이 포함됩니다. Exif 메타데이터의 `orientation` 필드에서 이미지의 방향을 찾을 수 있습니다. Amazon Rekognition Image는 Exif 메타데이터에서 이미지 방향 정보의 유무를 식별하지만

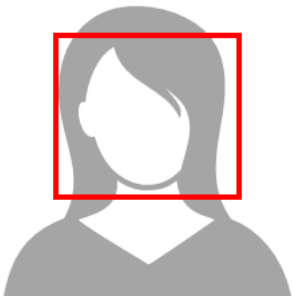
해당 정보에 대한 액세스를 제공하지 않습니다. 이미지의 Exif 메타데이터에 액세스하려면 타사 라이브러리를 사용하거나 고유한 코드를 작성하십시오. 자세한 내용은 Exif 버전 2.32 단원을 참조하십시오.

이미지의 방향을 알면 이미지를 회전하고 올바르게 표시하도록 코드를 작성할 수 있습니다.

경계 상자 표시

이미지 속의 얼굴을 분석하는 Amazon Rekognition Image 작업은 얼굴을 둘러싼 경계 상자의 좌표도 반환합니다. 자세한 내용을 알아보려면 BoundingBox를 참조하세요.

다음 이미지에 표시된 상자와 비슷한 경계 상자를 애플리케이션에서 얼굴 주위에 표시하려면 코드에 경계 상자 좌표를 사용하세요. 작업에서 반환된 경계 상자 좌표는 이미지의 방향을 반영합니다. 이미지를 올바르게 표시하기 위해 회전해야 하는 경우 경계 상자 좌표를 변환해야 할 수 있습니다.



Exif 메타데이터에 방향 정보가 있는 경우 경계 상자 표시

이미지의 방향이 Exif 메타데이터에 포함되어 있는 경우 Amazon Rekognition Image 작업은 다음을 수행합니다.

- 작업 응답의 방향 수정 필드에 null을 반환하십시오. 이미지를 회전하려면 코드에서 Exif 메타데이터에 제공된 방향을 사용하십시오.
- 이미 0도로 향한 경계 상자 좌표를 반환하십시오. 경계 상자를 올바른 위치에 표시하려면 반환된 좌표를 사용하십시오. 해당 좌표를 변환할 필요가 없습니다.

예제: 이미지 방향 및 이미지의 경계 상자 좌표 가져오기

다음 예제에서는 AWS SDK를 사용하여 Exif 이미지의 방향 데이터 및 `RecognizeCelebrities` 작업에서 감지된 유명 인사의 경계 상자 좌표를 가져오는 방법을 보여줍니다.

> **Note**
>
> `OrientationCorrection` 필드를 사용한 이미지 방향 추정 지원은 2021년 8월부로 중단되었습니다. API 응답에 포함된 해당 필드에 대해 반환되는 모든 값은 항상 NULL입니다.

Java

이 예제는 로컬 파일 시스템에서 이미지를 로드하고 `RecognizeCelebrities` 작업을 직접 호출하고 이미지의 높이와 너비를 결정하며 회전된 이미지에 대한 얼굴의 경계 상자 좌표를 계산합니다. 이 예제에서는 Exif 메타데이터에 저장된 방향 정보를 처리하는 방법을 보여주지 않습니다.

`main` 함수에서, `photo` 값을 로컬에 저장된 이미지(.png 또는 .jpg 형식)의 이름과 경로로 바꾸십시오.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 예제에서는 PIL/Pillow 이미지 라이브러리를 사용하여 이미지 너비와 높이를 확인합니다. 자세한 내용은 Pillow를 참조하십시오. 이 예제는 애플리케이션에서 필요할 수 있는 exif 메타데이터를 유지합니다.

`main` 함수에서, `photo` 값을 로컬에 저장된 이미지(.png 또는 .jpg 형식)의 이름과 경로로 바꾸십시오.

```
#Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

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

```
import java.util.List;

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

                Usage:    <sourceImage>

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

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

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

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

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

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

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

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

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

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

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

```
        // Calculate face position based on the image orientation.
        switch (rotation) {
            case "ROTATE_0" -> {
                left = imageWidth * box.left();
                top = imageHeight * box.top();
            }
            case "ROTATE_90" -> {
                left = imageHeight * (1 - (box.top() + box.height()));
                top = imageWidth * box.left();
            }
            case "ROTATE_180" -> {
                left = imageWidth - (imageWidth * (box.left() + box.width()));
                top = imageHeight * (1 - (box.top() + box.height()));
            }
            case "ROTATE_270" -> {
                left = imageHeight * box.top();
                top = imageWidth * (1 - box.left() - box.width());
            }
            default -> {
                System.out.println("No estimated orientation information. Check Exif
 data.");
                return;
            }
        }

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

저장된 비디오 분석 작업

Amazon Rekognition Video는 비디오를 분석하는 데 사용할 수 있는 API입니다. Amazon Rekognition Video를 사용하여 Amazon Simple Storage Service(S3) 버킷에 저장된 비디오에서 레이블, 얼굴, 사람, 유명 인사 및 성인(선정적이고 노골적인) 콘텐츠를 찾아낼 수 있습니다. 미디어 및 엔터테인먼트와 공공 안전 같은 카테고리에서 Amazon Rekognition Video를 사용할 수 있습니다. 예전에는 비디오를 스캔해 객체나 사람을 찾는 데 여러 시간이 걸렸으며, 사람의 육안 확인에 따른 실수도 잦았습니다. Amazon Rekognition Video는 동영상 전체에서 항목 및 발생 시기의 감지를 자동화합니다.

이 섹션에서는 Amazon Rekognition Video가 수행할 수 있는 분석 유형, API 개요, Amazon Rekognition Video 사용 예제를 다룹니다.

주제

- 분석 유형
- Amazon Rekognition Video API 개요
- Amazon Rekognition Video 작업 직접 호출
- Amazon Rekognition Video 구성
- Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)
- 를 사용하여 비디오 분석 AWS Command Line Interface
- 참조: 비디오 분석 결과 알림
- Amazon Rekognition Video 문제 해결

분석 유형

Amazon Rekognition Video를 사용하여 비디오에서 다음 정보를 분석할 수 있습니다.

- 비디오 세그먼트
- 레이블
- 선정적이고 노골적인 성인 콘텐츠
- 텍스트
- 유명 인사
- 얼굴
- 사람

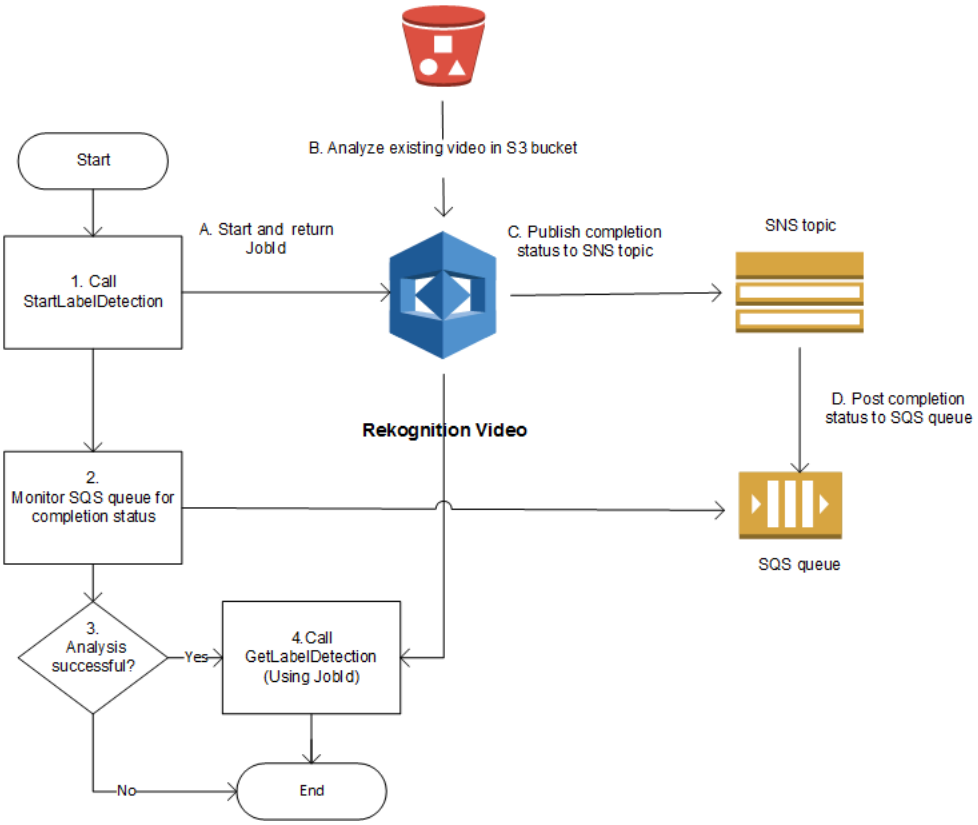
자세한 내용은 Amazon Rekognition 작동 방식 단원을 참조하십시오.

Amazon Rekognition Video API 개요

Amazon Rekognition Video는 Amazon S3 버킷에 저장된 비디오를 처리합니다. 디자인 패턴은 비동기 작업 세트입니다. StartLabelDetection과 같은 `Start` 작업을 직접 호출해 비디오 분석을 시작합니다. 동영상 분석 작업의 완료 상태는 Amazon Simple Notification Service(SNS) 주제에 게시됩니다. Amazon SNS 주제에서 완료 상태를 가져오려면 Amazon Simple Queue Service(Amazon SQS) 대기

열 또는 AWS Lambda 함수를 사용할 수 있습니다. 완료 상태가 되면 GetLabelDetection과 같은 Get 작업을 직접 호출해 요청 결과를 가져옵니다.

다음 다이어그램은 Amazon S3 버킷에 저장된 비디오에서 레이블을 감지하는 프로세스를 보여줍니다. 다이어그램에서 Amazon SQS 대기열은 Amazon SNS 주제의 완료 상태를 가져옵니다. 또는 AWS Lambda 함수를 사용할 수 있습니다.



이 프로세스는 다른 Amazon Rekognition Video 작업에서도 동일합니다. 다음 표에는 비스토리지 Amazon Rekognition 작업의 Start 및 Get 작업이 나열되어 있습니다.

탐지	시작 작업	가져오기 작업
비디오 세그먼트	StartSegmentDetection	GetSegmentDetection
레이블	StartLabelDetection	GetLabelDetection
노골적이거나 선정적인 성인 콘텐츠	StartContentModeration	GetContentModeration
텍스트	StartTextDetection	GetTextDetection

탐지	시작 작업	가져오기 작업
유명 인사	StartCelebrityRecognition	GetCelebrityRecognition
얼굴	StartFaceDetection	GetFaceDetection
사람	StartPersonTracking	GetPersonTracking

`GetCelebrityRecognition` 이외의 `Get` 작업의 경우 입력 비디오 전체에서 엔터티가 감지되는 시점의 추적 정보를 Amazon Rekognition Video가 반환합니다.

Amazon Rekognition Video에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 섹션을 참조하세요. Amazon SQS를 사용하여 비디오를 분석하는 예제는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 섹션을 참조하세요. AWS CLI 예제는 단원을 참조하십시오를 사용하여 비디오 분석 AWS Command Line Interface.

비디오 형식과 스토리지

Amazon Rekognition 작업으로 Amazon S3 버킷에 저장된 비디오를 분석할 수 있습니다. 비디오 분석 작업에 대한 모든 한도의 목록은 지침과 할당량 섹션을 참조하세요.

비디오는 H.264 코덱을 사용하여 인코딩해야 합니다. 지원되는 파일 형식은 MPEG-4와 MOV입니다.

코덱은 빠른 전달을 위해 데이터를 압축하고 수신한 데이터를 원래 형태로 압축을 해제하는 소프트웨어 또는 하드웨어입니다. H.264 코덱은 비디오 콘텐츠의 녹화, 압축 및 배포에 흔히 사용됩니다. 비디오 파일 형식에는 하나 이상의 코덱이 포함될 수 있습니다. MOV 또는 MPEG-4 형식의 비디오 파일이 Amazon Rekognition Video에서 작동되지 않을 경우에는 비디오 인코딩에 사용된 코덱이 H.264인지 확인하세요.

오디오 데이터를 분석하는 모든 Amazon Rekognition Video API는 AAC 오디오 코덱만 지원합니다.

저장된 비디오의 최대 파일 크기는 10GB입니다.

사람 검색

컬렉션에 저장된 얼굴 메타데이터를 사용하여 비디오에서 사람을 검색할 수 있습니다. 예를 들어, 보관된 비디오에서 특정 사람이나 여러 사람을 검색할 수 있습니다. IndexFaces 작업을 사용하여 컬렉션에 소스 이미지의 얼굴 메타데이터를 저장합니다. 그런 다음 StartFaceSearch를 사용하여 컬렉션에서 비동기 얼굴 검색을 시작할 수 있습니다. GetFaceSearch를 사용하여 검색 결과를 가져옵니다. 자

세한 내용은 저장된 비디오에서 얼굴 검색 단원을 참조하십시오. 사람 검색은 스토리지 기반 Amazon Rekognition 작업의 예입니다. 자세한 내용은 스토리지 기반 API 작업 단원을 참조하십시오.

스트리밍 비디오에서도 사람을 검색할 수 있습니다. 자세한 내용은 스트리밍 비디오 이벤트 작업 단원을 참조하십시오.

Amazon Rekognition Video 작업 직접 호출

Amazon Rekognition Video는 Amazon Simple Storage Service(S3) 버킷에 저장된 동영상을 분석하는 데 사용할 수 있는 비동기 API입니다. StartPersonTracking과 같은 Amazon Rekognition Video `Start` 작업을 직접 호출하여 비디오 분석을 시작합니다. Amazon Rekognition Video는 분석 요청의 결과를 Amazon Simple Notification Service(SNS) 주제에 게시합니다. Amazon Simple Queue Service(Amazon SQS) 대기열 또는 AWS Lambda 함수를 사용하여 Amazon SNS 주제에서 비디오 분석 요청의 완료 상태를 가져올 수 있습니다. 끝으로 GetPersonTracking과 같은 Amazon Rekognition `Get` 작업을 직접 호출하여 비디오 분석 요청 결과를 가져옵니다.

이후 섹션에서는 레이블 감지 작업을 사용하여 Amazon S3 버킷에 저장된 비디오에서 Amazon Rekognition Video가 레이블(객체, 이벤트, 개념 및 활동)을 감지하는 방법을 설명합니다. 동일한 접근 방식이 다른 Amazon Rekognition Video 작업(예: StartFaceDetection, StartPersonTracking)에도 적용됩니다. 예제 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)는 Amazon SQS 대기열을 사용하여 Amazon SNS 주제에서 완료 상태를 가져와 비디오를 분석하는 방법을 보여줍니다. 이것은 인물 경로 추적 같은 다른 Amazon Rekognition Video 예제의 기초로도 사용됩니다. AWS CLI 예제는 단원을 참조하십시오를 사용하여 비디오 분석 AWS Command Line Interface.

주제

- 비디오 분석 시작
- Amazon Rekognition Video 분석 요청의 완료 상태 가져오기
- Amazon Rekognition Video 분석 결과 가져오기

비디오 분석 시작

StartLabelDetection을 직접 호출하여 Amazon Rekognition Video 레이블 탐지 요청을 시작합니다. 다음은 `StartLabelDetection`으로 전달되는 JSON 요청의 예입니다.

```
{
    "Video": {
        "S3Object": {
```

```
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video.mp4"
        }
    },
    "ClientRequestToken": "LabelDetectionToken",
    "MinConfidence": 50,
    "NotificationChannel": {
        "SNSTopicArn": "arn:aws:sns:us-east-1:nnnnnnnnnn:topic",
        "RoleArn": "arn:aws:iam::nnnnnnnnnn:role/roleopic"
    },
    "JobTag": "DetectingLabels"
}
```

입력 파라미터 `Video`에는 비디오 파일 이름과 해당 파일을 가져올 Amazon S3 버킷이 있습니다. `NotificationChannel`에는 비디오 분석 요청이 완료될 때 Amazon Rekognition Video가 알려주는 Amazon SNS 주제의 Amazon 리소스 이름(ARN)이 있습니다. Amazon SNS 주제는 직접 호출할 Amazon Rekognition Video 엔드포인트와 동일한 AWS 리전에 있어야 합니다. `NotificationChannel`에는 Amazon Rekognition Video가 Amazon SNS 주제에 게시할 때 사용할 수 있는 역할의 ARN도 있습니다. IAM 서비스 역할을 생성하여 Amazon Rekognition이 Amazon SNS 주제에 게시할 수 있는 권한을 줍니다. 자세한 내용은 Amazon Rekognition Video 구성 단원을 참조하십시오.

Amazon SNS 주제에 게시된 완료 상태에서 작업을 식별할 때 사용할 수 있는 선택사항인 입력 파라미터인 `JobTag`도 지정할 수 있습니다.

분석 작업이 실수로 중복되지 않도록 idempotent 토큰 `ClientRequestToken`을 선택적으로 제공할 수 있습니다. `ClientRequestToken` 값을 제공하면 `Start` 작업에서 `StartLabelDetection` 같은 시작 작업의 여러 동일한 호출에 대해 동일한 `JobId`를 반환합니다. `ClientRequestToken` 토큰은 수명이 7일입니다. 7일 후에 다시 사용할 수 있습니다. 토큰 수명 기간 동안 토큰을 재사용할 경우 다음과 같은 현상이 생깁니다.

- 동일한 `Start` 작업과 동일한 입력 파라미터로 토큰을 다시 사용할 경우 동일한 `JobId`가 반환됩니다. 이 작업은 다시 실행되지 않으며 Amazon Rekognition Video는 등록된 Amazon SNS 주제로 완료 상태를 전송하지 않습니다.
- 동일한 `Start` 작업에서 약간의 입력 파라미터를 변경하여 토큰을 재사용할 경우 `IdempotentParameterMismatchException`(HTTP 상태 코드: 400) 예외가 표시됩니다.
- Amazon Rekognition에서 예기치 않은 결과를 얻을 수 있으므로 다른 `Start` 작업으로 토큰을 재사용하지 않아야 합니다.

StartLabelDetection 작업에 대한 응답은 작업 식별자입니다(JobId). Amazon Rekognition Video가 Amazon SNS 주제에 완료 상태를 게시한 후에 JobId를 사용하여 요청을 추적하고 분석 결과를 가져오세요. 예제:

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

너무 많은 작업을 동시에 시작하면서 StartLabelDetection을 직접 호출할 경우에는 동시 실행 작업의 수가 Amazon Rekognition 서비스 제한 미만이 될 때까지 LimitExceededException(HTTP 상태 코드: 400)이 발생합니다.

다수의 작업으로 인해 LimitExceededException 예외가 발생하는 경우에는 Amazon SQS 대기열을 사용하여 수신되는 요청을 관리하는 것이 좋습니다. Amazon SQS 대기열에서 평균 동시 요청 수를 관리할 수 없고 여전히 LimitExceededException 예외가 발생하는 경우 AWS 지원팀에 문의하세요.

Amazon Rekognition Video 분석 요청의 완료 상태 가져오기

Amazon Rekognition Video는 등록된 Amazon SNS 주제로 분석 완료 알림을 보냅니다. 알림에는 JSON 문자열 형태의 작업 완료 상태와 작업 식별자가 포함됩니다. 성공적인 비디오 분석 요청은 SUCCEEDED 상태를 갖습니다. 예를 들어, 다음 결과는 레이블 감지 작업의 성공적인 처리를 보여줍니다.

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1nnnnnnnnnnn",
    "Status": "SUCCEEDED",
    "API": "StartLabelDetection",
    "JobTag": "DetectingLabels",
    "Timestamp": 1510865364756,
    "Video": {
        "S3ObjectName": "video.mp4",
        "S3Bucket": "amzn-s3-demo-bucket"
    }
}
```

자세한 내용은 참조: 비디오 분석 결과 알림 단원을 참조하십시오.

Amazon Rekognition Video가 Amazon SNS 주제에 게시한 상태 정보를 가져오려면 다음 옵션 중 하나를 사용합니다.

- AWS Lambda - Amazon SNS 주제에 기록하는 AWS Lambda 함수를 구독할 수 있습니다. Amazon Rekognition이 Amazon SNS 주제에 요청 완료 사실을 알릴 때 이 함수가 직접 호출됩니다. 서버측

코드로 비디오 분석 요청 결과를 처리해야 할 경우에 Lambda 함수를 사용합니다. 예를 들어, 서버 측 코드를 사용하여 비디오에 주석을 달거나 비디오 콘텐츠에 관한 보고서를 생성한 후에 정보를 클라이언트 애플리케이션으로 반환해야 하는 경우가 있을 수 있습니다. Amazon Rekognition API가 대용량 데이터를 반환할 수 있으므로 대용량 비디오는 서버측 처리를 권장합니다.

- Amazon Simple Queue Service - Amazon SQS 대기열에서 Amazon SNS 주제를 구독할 수 있습니다. 그런 다음 Amazon SQS 대기열을 폴링하여 비디오 분석 요청이 완료될 때 Amazon Rekognition이 게시하는 완료 상태를 가져옵니다. 자세한 내용은 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 단원을 참조하십시오. Amazon Rekognition Video 작업을 클라이언트 애플리케이션에서만 직접 호출해야 할 경우에는 Amazon SQS 대기열을 사용합니다.

> **Important**
>
> Amazon Rekognition Video `Get` 작업을 반복해서 직접 호출하여 요청 완료 상태를 가져오는 것은 권장하지 않습니다. 그 이유는 요청이 너무 많을 경우 Amazon Rekognition Video가 `Get` 작업을 제한하기 때문입니다. 여러 비디오를 동시에 처리할 경우 각 비디오의 상태를 개별적으로 확인하기 위해 Amazon Rekognition Video를 폴링하는 것보다는 하나의 SQS 대기열을 보고 완료 알림을 모니터링하는 것이 더 효율적이고 간단합니다.

Amazon Rekognition Video 분석 결과 가져오기

비디오 분석 요청의 결과를 가져오려면 먼저 Amazon SNS 주제에서 검색된 완료 상태가 `SUCCEEDED`인지 확인합니다. 그런 다음 `GetLabelDetection`을 호출하면 이에 의해 `StartLabelDetection`에서 반환된 `JobId` 값이 통과됩니다. 요청 JSON은 다음 예제와 비슷합니다.

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

JobId는 비디오 분석 작업용 ID입니다. 비디오 분석 시 대용량 데이터가 생성될 수 있으므로 `MaxResults`를 사용하여 단일 Get 작업에서 반환할 결과의 최대수를 지정합니다. `MaxResults`의 기본값은 1000입니다. 1,000보다 큰 값을 지정한 경우 최대 1,000개의 결과가 반환됩니다. 작업에서 전체 결과 세트가 반환되지 않을 경우에는 그 다음 페이지의 페이지 매김 토큰이 작업 응답으로 반환됩니

다. 이전의 Get 요청에서 페이지 매김 토큰을 받은 경우에는 이것을 NextToken과 함께 사용하여 결과의 다음 페이지를 가져옵니다.

> **Note**
>
> Amazon Rekognition은 비디오 분석 작업의 결과를 7일 동안 유지합니다. 이 후에는 분석 결과를 검색하지 못합니다.

GetLabelDetection 작업 응답 JSON은 다음과 비슷합니다.


```
{
    "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 및 GetContentModeration 연산을 통해 타임스탬프 또는 레이블 이름을 기준으로 분석 결과를 정렬할 수 있습니다. 비디오 세그먼트 또는 타임스탬프 기준으로도 결과를 집계할 수 있습니다.

결과를 감지 시간별로(비디오 시작 후 경과한 밀리초) 또는 감지 엔터티(객체, 얼굴, 유명 인사, 중재 레이블 또는 사람)의 알파벳 순으로 정렬할 수 있습니다. 시간별로 정렬하려면 SortBy 입력 파라미터 값을 TIMESTAMP로 설정합니다. SortBy가 지정되지 않으면 기본적으로 시간별로 정렬됩니다. 앞의 예제는 시간별로 정렬된 경우입니다. 엔터티별로 정렬하려면 SortBy 입력 파라미터를 수행할 작업에 해당하는 값과 함께 사용합니다. 예를 들어, GetLabelDetection 호출에서 감지된 레이블별로 정렬하려면 NAME 값을 사용합니다.

타임스탬프 기준으로 결과를 집계하려면 AggregateBy 파라미터 값을 TIMESTAMPS로 설정합니다. 비디오 세그먼트별로 집계하려면 AggregateBy의 값을 SEGMENTS로 설정합니다. SEGMENTS 집계 모드는 시간 경과에 따라 레이블을 집계하는 반면 TIMESTAMPS는 2FPS 샘플링과 프레임당 출력을 사용하여 레이블이 감지된 타임스탬프를 제공합니다(참고: 현재 샘플링 속도는 변경될 수 있으므로 현재 샘플링 속도를 가정해서는 안 됨). 값을 지정하지 않을 경우 기본 집계 방식은 TIMESTAMPS입니다.

Amazon Rekognition Video 구성

저장된 비디오에 Amazon Rekognition Video API를 사용하려면 Amazon SNS 주제에 액세스하도록 사용자 및 IAM 서비스 역할을 구성해야 합니다. 또한 Amazon SNS 주제에 대한 Amazon SQS 대기열을 구독해야 합니다.

> **Note**
>
> Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 예제를 설정하기 위해 이 지침을 사용하는 경우에는 3, 4, 5 및 6단계를 수행할 필요가 없습니다. 이 예제에는 Amazon SNS 주제 및 Amazon SQS 대기열을 생성하고 구성하기 위한 코드가 포함되어 있습니다.

이 섹션의 예제에서는 Amazon Rekognition Video가 여러 개의 주제에 액세스할 수 있는 권한을 부여하는 지침을 사용하여 새로운 Amazon SNS 주제를 생성합니다. 기존 Amazon SNS 주제를 사용하고자 하는 경우 3단계에서 기존 Amazon SNS 주제에 대한 액세스 권한 부여을 사용합니다.

Amazon Rekognition Video를 구성하려면

1. Amazon Rekognition Video에 액세스하도록 AWS 계정을 설정합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.
2. 필요한 AWS SDK를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
3. 이 개발자 안내서의 코드 예제를 실행하려면 선택한 사용자에게 프로그래밍 방식 액세스 권한이 있어야 합니다. 자세한 정보는 프로그래밍 방식 액세스 권한 부여을 참조하세요.

   또한 사용자에게는 최소한 다음과 같은 권한이 필요합니다.

   - AmazonSQSFullAccess
   - AmazonRekognitionFullAccess
   - AmazonS3FullAccess
   - AmazonSNSFullAccess

   IAM Identity Center를 사용하여 인증하는 경우 사용자 역할의 권한 세트에 해당 권한을 추가하고, 그렇지 않은 경우 사용자의 IAM 역할에 권한을 추가하세요.
4. Amazon SNS 콘솔을 사용하여 Amazon SNS 주제를 생성합니다. 주제 이름에 AmazonRekognition을 추가합니다. 주제 Amazon Resource Name(ARN)을 기록합니다. 사용하는 AWS 엔드포인트와 동일한 리전에 주제가 있어야 합니다.
5. Amazon SQS 콘솔을 사용하여 Amazon SQS 표준 대기열을 생성합니다. 대기열 ARN을 기록합니다.
6. 3단계에서 만든 주제에 대한 대기열을 구독합니다.
7. Amazon SQS 대기열에 메시지를 전송하도록 Amazon SNS 주제에 권한을 부여합니다.
8. Amazon Rekognition Video가 Amazon SNS 주제에 액세스할 권한을 부여할 IAM 서비스 역할을 생성합니다. 서비스 역할의 Amazon Resource Name(ARN)을 적어둡니다. 자세한 내용은 여러 Amazon SNS 주제에 대한 액세스 권한 부여 단원을 참조하십시오.
9. 계정 보안을 유지하려면 Rekognition의 액세스 범위를 사용 중인 리소스로만 제한합니다. 이는 IAM 서비스 역할에 신뢰 정책을 추가하는 것으로 수행할 수 있습니다. 이렇게 하는 방법에 대한 정보는 교차 서비스 혼동된 대리인 방지 단원을 참조하십시오.
10. 1단계에서 생성한 사용자에게 다음 인라인 정책을 추가합니다.

    인라인 정책에 이름을 지정합니다.

11. 고객 관리형 AWS Key Management Service 키를 사용하여 Amazon S3 버킷의 비디오를 암호화하는 경우 7단계에서 생성한 서비스 역할이 비디오를 해독할 수 있도록 키에 권한을 추가합니다. 서비스 역할에는 최소한 `kms:GenerateDataKey` 및 `kms:Decrypt` 작업에 대한 권한이 필요합니다. 예제:

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

    자세한 정보는 내 Amazon S3 버킷은 사용자 지정 AWS KMS 키를 사용하는 기본 암호화를 사용합니다. 사용자가 버킷에서 다운로드하고 버킷에 업로드하도록 허용하려면 어떻게 해야 합니까? 및 AWS Key Management Service에 저장된 KMS 키를 사용한 서버측 암호화(SSE-KMS)를 사용하여 데이터 보호를 참조하세요.

12. 이제 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 및 를 사용하여 비디오 분석 AWS Command Line Interface에서 예시를 실행할 수 있습니다.

여러 Amazon SNS 주제에 대한 액세스 권한 부여

Amazon Rekognition Video가 사용자가 생성하는 Amazon SNS 주제에 액세스할 권한을 부여하기 위해서는 IAM 서비스 역할을 사용합니다. IAM은 Amazon Rekognition Video 서비스 역할을 생성하기 위한 Rekognition 사용 사례를 제공합니다.

`AmazonRekognitionServiceRole` 권한 정책을 사용하고 주제 이름 앞에 AmazonRekognition을 추가하여 Amazon Rekognition Video에 여러 Amazon SNS 주제에 대한 액세스 권한을 부여할 수 있습니다(예: `AmazonRekognitionMyTopicName`).

Amazon Rekognition Video에 여러 Amazon SNS 주제에 대한 액세스 권한을 부여하려면

1. IAM 서비스 역할을 생성합니다. 다음 정보를 사용하여 IAM 서비스 역할을 생성하세요.

    1. 서비스 이름으로 Rekognition을 선택합니다.

2. 서비스 역할 사용 사례로 [Rekognition]을 선택합니다. 나열된 AmazonRekognitionServiceRole 권한 정책을 확인합니다. AmazonRekognitionServiceRole은 Amazon Rekognition Video에 AmazonRekognition 접두사가 붙은 Amazon SNS 주제에 대한 액세스 권한을 부여합니다.
3. 서비스 역할에 이름을 지정합니다.

2. 서비스 역할의 ARN을 기록합니다. 이것은 비디오 분석 작업을 시작할 때 필요합니다.

기존 Amazon SNS 주제에 대한 액세스 권한 부여

Amazon Rekognition Video가 기존 Amazon SNS 주제에 액세스할 수 있도록 허용하는 권한 정책을 만들 수 있습니다.

Amazon Rekognition Video에 기존 Amazon SNS 주제에 대한 액세스 권한을 부여하려면

1. IAM JSON 정책 편집기로 새 권한 정책을 생성하고 다음 정책을 사용합니다. `topicarn`을 원하는 Amazon SNS 주제의 Amazon 리소스 이름(ARN)으로 바꿉니다.
2. IAM 서비스 역할을 생성하거나 기존 IAM 서비스 역할을 업데이트합니다. 다음 정보를 사용하여 IAM 서비스 역할을 생성하세요.
   1. 서비스 이름으로 Rekognition을 선택합니다.
   2. 서비스 역할 사용 사례로 [Rekognition]을 선택합니다.
   3. 1단계에 만든 권한 정책을 연결합니다.
3. 서비스 역할의 ARN을 기록합니다. 이것은 비디오 분석 작업을 시작할 때 필요합니다.

Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)

이 절차는 Amazon Rekognition Video 레이블 감지 작업, Amazon S3 버킷에 저장된 비디오, 그리고 Amazon SNS 주제를 사용하여 비디오에서 레이블을 감지하는 방법을 보여줍니다. 또한 Amazon SQS 대기열을 사용하여 Amazon SNS 주제에서 완료 상태를 가져오는 방법도 보여줍니다. 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오. Amazon SQS 대기열 사용에는 제한이 없습니다. 예를 들어 AWS Lambda 함수를 사용하여 완료 상태를 가져올 수 있습니다. 자세한 내용은 Amazon SNS 알림을 사용하여 Lambda 함수 호출 단원을 참조하십시오.

이 절차의 예제 코드는 다음을 실행하는 방법을 보여줍니다.

1. Amazon SNS 주제를 생성합니다.
2. Amazon SQS 대기열을 생성합니다.

3. Amazon Rekognition Video에 비디오 분석 작업의 완료 상태를 Amazon SNS 주제에 게시할 권한을 부여합니다.
4. Amazon SQS 대기열에서 Amazon SNS 주제를 구독합니다.
5. StartLabelDetection을 직접 호출하여 비디오 분석 요청을 시작합니다.
6. Amazon SQS 대기열에서 완료 상태를 가져옵니다. 이 예제는 `StartLabelDetection`에서 반환되는 작업 식별자(`JobId`)를 추적하여 완료 상태에서 판독되는 작업 식별자와 일치하는 결과만 가져옵니다. 이점은 다른 애플리케이션에서 동일한 대기열과 주제를 사용할 경우에 중요하게 고려해야 합니다. 간소화를 위해 이 예제에서는 일치하지 않는 작업을 삭제합니다. 그러한 작업을 Amazon SQS DLQ(Dead Letter Queue)에 추가하여 추후 조사하는 것을 고려해 보세요.
7. GetLabelDetection을 직접 호출하여 비디오 분석 결과를 가져오고 표시합니다.

사전 조건

이 절차에 대한 예제 코드는 Java 및 Python에 제공됩니다. 적절한 AWS SDK가 설치되어 있어야 합니다. 자세한 내용은 Amazon Rekognition 시작 단원을 참조하십시오. 사용할 AWS 계정은 Amazon Rekognition API에 대한 액세스 권한을 가지고 있어야 합니다. 자세한 내용은 Amazon Rekognition에서 정의한 작업을 참조하세요.

비디오에서 레이블을 감지하려면

1. Amazon Rekognition Video에 대한 사용자 액세스를 구성하고 Amazon Rekognition Video에서 Amazon SNS에 액세스할 수 있도록 구성합니다. 자세한 내용은 Amazon Rekognition Video 구성 단원을 참조하십시오. 예제 코드에서 Amazon SNS 주제 및 Amazon SQS 대기열을 생성하고 구성하기 때문에 3, 4, 5 및 6단계를 수행할 필요가 없습니다.
2. MOV 또는 MPEG-4 형식 비디오 파일을 Amazon S3 버킷으로 업로드합니다. 테스트할 때는 30초 이하의 비디오를 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 코드 예제를 사용하여 비디오에서 레이블을 감지합니다.

   Java

   `main` 함수에서 수행:

   - `roleArn`을 Amazon Rekognition Video를 구성하려면의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 바꿉니다.

- amzn-s3-demo-bucket 및 video 값을 2단계에서 지정한 버킷과 비디오 파일 이름으로 바꿉니다.

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

package com.amazonaws.samples;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Condition;
import com.amazonaws.auth.policy.Principal;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.SQSActions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CelebrityDetail;
import com.amazonaws.services.rekognition.model.CelebrityRecognition;
import com.amazonaws.services.rekognition.model.CelebrityRecognitionSortBy;
import com.amazonaws.services.rekognition.model.ContentModerationDetection;
import com.amazonaws.services.rekognition.model.ContentModerationSortBy;
import com.amazonaws.services.rekognition.model.Face;
import com.amazonaws.services.rekognition.model.FaceDetection;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.FaceSearchSortBy;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.GetContentModerationRequest;
import com.amazonaws.services.rekognition.model.GetContentModerationResult;
import com.amazonaws.services.rekognition.model.GetFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.GetFaceDetectionResult;
import com.amazonaws.services.rekognition.model.GetFaceSearchRequest;
import com.amazonaws.services.rekognition.model.GetFaceSearchResult;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
import com.amazonaws.services.rekognition.model.GetPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.GetPersonTrackingResult;
import com.amazonaws.services.rekognition.model.Instance;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.services.rekognition.model.LabelDetection;
import com.amazonaws.services.rekognition.model.LabelDetectionSortBy;
```

```
import com.amazonaws.services.rekognition.model.NotificationChannel;
import com.amazonaws.services.rekognition.model.Parent;
import com.amazonaws.services.rekognition.model.PersonDetection;
import com.amazonaws.services.rekognition.model.PersonMatch;
import com.amazonaws.services.rekognition.model.PersonTrackingSortBy;
import com.amazonaws.services.rekognition.model.S3Object;
import
 com.amazonaws.services.rekognition.model.StartCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.StartCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.StartContentModerationRequest;
import com.amazonaws.services.rekognition.model.StartContentModerationResult;
import com.amazonaws.services.rekognition.model.StartFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.StartFaceDetectionResult;
import com.amazonaws.services.rekognition.model.StartFaceSearchRequest;
import com.amazonaws.services.rekognition.model.StartFaceSearchResult;
import com.amazonaws.services.rekognition.model.StartLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.StartLabelDetectionResult;
import com.amazonaws.services.rekognition.model.StartPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.StartPersonTrackingResult;
import com.amazonaws.services.rekognition.model.Video;
import com.amazonaws.services.rekognition.model.VideoMetadata;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.CreateTopicResult;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.*;

public class VideoDetect {


    private static String sqsQueueName=null;
    private static String snsTopicName=null;
    private static String snsTopicArn = null;
    private static String roleArn= null;
    private static String sqsQueueUrl = null;
    private static String sqsQueueArn = null;
```

```
    private static String startJobId = null;
    private static String bucket = null;
    private static String video = null;
    private static AmazonSQS sqs=null;
    private static AmazonSNS sns=null;
    private static AmazonRekognition rek = null;

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


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

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

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

        CreateTopicandQueue();

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

        StartLabelDetection(bucket, video);

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

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


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

    }


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

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

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

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

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

```
                        }


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

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

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

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


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

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


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

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

    }

    private static void GetLabelDetectionResults() throws Exception{

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

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

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


            labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

            VideoMetadata videoMetaData=labelDetectionResult.getVideoMetadata();

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


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

            for (LabelDetection detectedLabel: detectedLabels) {
                long seconds=detectedLabel.getTimestamp();
                Label label=detectedLabel.getLabel();
```

```
                System.out.println("Millisecond: " + Long.toString(seconds) + "
");

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

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

   }

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

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

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

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

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


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


        System.out.println("Topic arn: " + snsTopicArn);
        System.out.println("Queue arn: " + sqsQueueArn);
        System.out.println("Queue url: " + sqsQueueUrl);
        System.out.println("Queue sub arn: " + sqsSubscriptionArn );
    }
   static void DeleteTopicandQueue()
   {
```

```
        if (sqs !=null) {
            sqs.deleteQueue(sqsQueueUrl);
            System.out.println("SQS queue deleted");
        }

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

Python

main 함수에서 수행:

- roleArn을 Amazon Rekognition Video를 구성하려면의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 바꿉니다.
- amzn-s3-demo-bucket 및 video 값을 2단계에서 지정한 버킷과 비디오 파일 이름으로 바꿉니다.
- Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.
- 설정 파라미터에 필터링 기준을 포함할 수도 있습니다. 예를 들어, LabelsInclusionFilter 또는 LabelsExclusionFilter를 원하는 값의 목록과 함께 사용할 수 있습니다. 아래 코드에서 Features and Settings 섹션의 주석을 제거하고 사용자가 원하는 값을 제공하여 반환되는 결과를 관심 있는 레이블로만 제한할 수 있습니다.
- GetLabelDetection을 직접 호출할 때 SortBy 및 AggregateBy 인수의 값을 제공할 수 있습니다. 시간별로 정렬하려면 SortBy 입력 파라미터 값을 TIMESTAMP로 설정합니다. 엔터티별로 정렬하려면 SortBy 입력 파라미터를 수행할 작업에 해당하는 값과 함께 사용합니다. 타임스탬프 기준으로 결과를 집계하려면 AggregateBy 파라미터 값을 TIMESTAMPS로 설정합니다. 비디오 세그먼트 기준으로 집계하려면 SEGMENTS를 사용합니다.

코드 예제에서 줄 policy = """""".format(sqsQueueArn, self.snsTopicArn)의 따옴표 안에 다음 IAM 정책을 배치합니다.

```
## Copyright 2018 Amazon.com, Inc. or its 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
        # Insert the policy from the documentation from
        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

다음 예제 코드에서 이렇게 하세요.

- REGION의 값을 계정의 운영 리전 이름으로 바꿉니다.
- amzn-s3-demo-bucket의 값을 비디오 파일이 들어 있는 Amazon S3 버킷의 이름으로 바꿉니다.
- videoName의 값을 Amazon S3 버킷에 들어 있는 비디오 파일 이름으로 바꿉니다.
- Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.
- roleArn을 Amazon Rekognition Video를 구성하려면의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 바꿉니다.

```
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",&TCX5-2025-waiver;
     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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

            if (status.compareTo("SUCCEEDED") == 0)
```

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

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

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

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

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

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

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

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

                // Get the status and job id from the notification
                ObjectMapper mapper = new ObjectMapper();
                JsonNode jsonMessageTree = mapper.readTree(notification);
                JsonNode messageBodyText = jsonMessageTree.get("Message");
                ObjectMapper operationResultMapper = new ObjectMapper();
                JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                JsonNode operationJobId = jsonResultTree.get("JobId");
                JsonNode operationStatus = jsonResultTree.get("Status");
                System.out.println("Job found in JSON is " + operationJobId);

                DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
```

```
                    .queueUrl(queueUrl)
                    .build();

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

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

                    sqs.deleteMessage(deleteMessageRequest);
                }

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

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

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

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

    try {
        do {
```

```
            if (labelDetectionResult !=null)
                paginationToken = labelDetectionResult.nextToken();


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

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

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

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

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

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

```
                System.out.println("   Parent labels for " + label.name() +
":");
                List<Parent> parents = label.parents();

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

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

4. 코드를 작성하고 실행합니다. 이 작업은 마치는 데 시간이 걸릴 수 있습니다. 작업이 끝나면 비디오에서 감지된 레이블 목록이 표시됩니다. 자세한 내용은 비디오에서 레이블 감지 단원을 참조하십시오.

를 사용하여 비디오 분석 AWS Command Line Interface

AWS Command Line Interface (AWS CLI)를 사용하여 Amazon Rekognition Video 작업을 호출할 수 있습니다. 설계 패턴은 AWS SDK for Java 또는 다른 AWS SDKs와 함께 Amazon Rekognition Video API를 사용하는 것과 동일합니다. 자세한 내용은 Amazon Rekognition Video API 개요 단원을 참조하십시오. 다음 절차에서는를 사용하여 비디오에서 레이블을 감지 AWS CLI 하는 방법을 보여줍니다.

`start-label-detection`을 불러와 비디오에서 레이블 감지을 시작합니다. Amazon Rekognition이 비디오 분석을 마치면 `start-label-detection`의 `--notification-channel` 파라미터에 지정된 Amazon SNS 주제로 그 완료 상태가 전송됩니다. 완료 상태를 가져오려면 Amazon Simple Queue Service(Amazon SQS) 대기열에서 Amazon SNS 주제를 구독하면 됩니다. 그런 다음 receive-message를 폴링하여 Amazon SQS 대기열에서 완료 상태를 가져옵니다.

StartLabelDetection을 직접 호출할 때 LabelsInclusionFilter 및/또는 LabelsExclusionFilter 인수에 필터링 인수를 제공하여 결과를 필터링할 수 있습니다. 자세한 정보는 비디오에서 레이블 감지을 참조하십시오.

완료 상태 알림은 receive-message 응답 내의 JSON 구조입니다. 응답에서 JSON을 추출해야 합니다. 완료 상태 JSON에 관한 내용은 참조: 비디오 분석 결과 알림 단원을 참조하십시오. 완료 상태 JSON의 Status 필드 값이 SUCCEEDED이면 get-label-detection을 불러와 비디오 분석 요청의 결과를 가져올 수 있습니다. GetLabelDetection을 직접 호출할 때 SortBy 및 AggregateBy 인수를 사용하여 반환된 결과를 정렬하고 집계할 수 있습니다.

다음 절차에는 Amazon SQS 대기열을 폴링하는 코드가 포함되어 있지 않습니다. 또한 Amazon SQS 대기열에서 반환되는 JSON의 구문 분석용 코드도 포함되지 않습니다. Java 예제는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 단원을 참조하십시오.

사전 조건

이 절차를 실행하려면가 AWS CLI 설치되어 있어야 합니다. 자세한 내용은 Amazon Rekognition 시작 단원을 참조하십시오. 사용할 AWS 계정은 Amazon Rekognition API에 대한 액세스 권한을 가지고 있어야 합니다. 자세한 내용은 Amazon Rekognition에서 정의한 작업을 참조하세요.

Amazon Rekognition Video를 구성하고 비디오를 업로드하려면

1. Amazon Rekognition Video에 대한 사용자 액세스를 구성하고 Amazon Rekognition Video에서 Amazon SNS에 액세스할 수 있도록 구성합니다. 자세한 내용은 Amazon Rekognition Video 구성 단원을 참조하십시오.
2. MOV 또는 MPEG-4 형식 비디오 파일을 S3 버킷으로 업로드합니다. 개발 및 테스트를 진행할 때는 30초 이하의 짧은 비디오를 사용하는 것이 좋습니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

비디오에서 레이블을 감지하려면

1. 다음 AWS CLI 명령을 실행하여 비디오에서 레이블 감지를 시작합니다.

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

```
--profile profile-name \
--settings "{"GeneralLabels":{"LabelInclusionFilters":["Car"]}}
```

다음 값을 업데이트합니다.

- `amzn-s3-demo-bucket` 및 `videofile`을 2단계에서 지정한 Amazon S3 버킷 이름과 파일 이름으로 변경합니다.
- `us-east-1`을 사용 중인 AWS 리전으로 변경합니다.
- Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.
- `TopicARN`을 Amazon Rekognition Video 구성의 3단계에서 생성한 Amazon SNS 주제의 ARN으로 변경합니다.
- `RoleARN`을 Amazon Rekognition Video 구성의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 변경합니다.
- 필요한 경우 `endpoint-url`을 지정할 수 있습니다. AWS CLI는 제공된 리전을 기반으로 적절한 엔드포인트 URL을 자동으로 결정할 것입니다. 하지만 프라이빗 VPC의 엔드포인트를 사용하는 경우에는 `endpoint-url`을 지정해야 할 수 있습니다. AWS Service 엔드포인트 리소스에는 엔드포인트 URL을 지정하는 구문과 각 리전의 이름 및 코드가 나열되어 있습니다.
- 설정 파라미터에 필터링 기준을 포함할 수도 있습니다. 예를 들어, `LabelsInclusionFilter` 또는 `LabelsExclusionFilter`를 원하는 값의 목록과 함께 사용할 수 있습니다.

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예시를 보려면 다음을 참조하세요.

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

2. 응답의 `JobId` 값을 기록합니다. 이 응답은 다음 JSON 예제와 비슷해 보입니다.

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

```
}
```

3. Amazon SQS 대기열을 폴링하여 완료 상태 JSON을 확인하는 코드를 작성합니다(receive-message 사용).
4. 코드를 적어 완료 상태 JSON에서 Status 필드를 추출합니다.
5. 값이 Status인 경우 다음 AWS CLI 명령을 SUCCEEDED실행하여 레이블 감지 결과를 표시합니다.

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

다음 값을 업데이트합니다.

- JobId를 변경하여 2단계에서 기록한 작업 식별자와 일치시킵니다.
- Endpoint 및 us-east-1을 사용하는 AWS 엔드포인트와 리전으로 변경합니다.

그 결과는 다음 예제 JSON과 비슷해 보입니다.

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

참조: 비디오 분석 결과 알림

Amazon Rekognition은 완료 상태를 포함한 Amazon Rekognition Video 분석 요청의 결과를 Amazon Simple Notification Service(SNS) 주제에 게시합니다. Amazon SNS 주제에서 알림을 받으려면 Amazon Simple Queue Service 대기열 또는 AWS Lambda 함수를 사용합니다. 자세한 내용은 the section called “Amazon Rekognition Video 작업 직접 호출” 단원을 참조하십시오. 예제는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 섹션을 참조하세요.

페이로드는 다음 JSON 형식입니다.

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

이름	설명
JobId	작업 식별자입니다. StartPersonTracking과 같은 `Start` 작업에서 반환되는 작업 식별자와 일치합니다.
Status	작업의 상태입니다. 유효 값은 SUCCEEDED, FAILED 또는 ERROR입니다.

이름	설명
API	입력 비디오를 분석할 때 사용되는 Amazon Rekognition Video 작업입니다.
JobTag	작업의 식별자입니다. StartLabelDetection과 같은 Start 작업을 직접 호출할 때 JobTag를 지정합니다.
타임스탬프	작업이 완료된 시점의 Unix 타임스탬프입니다.
비디오	처리된 비디오에 관한 세부 정보입니다. 파일 이름과 파일이 저장된 Amazon S3 버킷이 포함되어 있습니다.

다음은 Amazon SNS 주제로 전송된 성공적인 알림의 예입니다.

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

Amazon Rekognition Video 문제 해결

다음은 Amazon Rekognition Video 및 저장된 비디오 작업 시 문제 해결 정보를 다룹니다.

Amazon SNS 주제로 전송된 완료 상태를 수신할 수 없습니다

Amazon Rekognition Video는 비디오 분석이 완료되면 상태 정보를 Amazon SNS 주제에 게시합니다. 일반적으로 Amazon SQS 대기열 또는 Lambda 함수를 통해 주제를 구독함으로써 완료 상태 메시지를 받습니다. 조사에 도움이 되도록 이메일로 Amazon SNS 주제를 구독하세요. 그러면 Amazon SNS 주

제로 전송된 메시지를 이메일 수신함에서 받으실 수 있습니다. 자세한 내용은 Amazon SNS 주제에 구독 설정을 참조하세요.

애플리케이션에서 메시지를 수신하지 않는 경우 다음을 고려하십시오.

- 분석이 완료되었는지 확인합니다. Get 작업 응답(예: `GetLabelDetection`)의 `JobStatus` 값을 확인합니다. 값이 `IN_PROGRESS`인 경우 분석이 끝난 것이 아닙니다. 따라서 완료 상태도 아직 Amazon SNS 주제에 게시되지 않았습니다.
- Amazon Rekognition Video이 Amazon SNS 주제에 게시할 권한을 부여하는 IAM 서비스 역할이 있는지 확인합니다. 자세한 내용은 Amazon Rekognition Video 구성 단원을 참조하십시오.
- 사용 중인 IAM 서비스 역할이 역할 보안 인증을 사용하여 Amazon SNS 주제에 게시할 수 있고 서비스 역할의 권한 범위가 사용 중인 리소스로 안전하게 지정되어 있는지 확인하세요. 다음 단계를 수행합니다.
  - 사용자의 Amazon Resource Name(ARN)을 가져옵니다.

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

  - 역할 신뢰 관계에 사용자 ARN을 추가합니다. 자세한 내용은 역할 수정을 참조하세요. 다음 예제 신뢰 정책은 사용자의 역할 보안 인증 정보를 지정하고 서비스 역할의 권한을 사용 중인 리소스로만 제한합니다(서비스 역할의 권한 범위를 안전하게 제한하는 방법에 대한 자세한 내용은 교차 서비스 혼동된 대리인 방지 참조).
  - 역할 수임: `aws sts assume-role --role-arn arn:Role ARN --role-session-name SessionName --profile RekognitionUser`
  - Amazon SNS 주제에 게시: `aws sns publish --topic-arn arn:Topic ARN --message "Hello World!" --region us-east-1 --profile RekognitionUser`

  AWS CLI 명령이 작동하면 메시지를 받게 됩니다(이메일로 주제를 구독한 경우 이메일 받은 편지함에서). 메시지를 수신하지 못한 경우:

  - Amazon Rekognition Video를 구성했는지 확인합니다. 자세한 내용은 Amazon Rekognition Video 구성 단원을 참조하십시오.
  - 이 문제 해결 질문에 대한 기타 팁을 확인합니다.
- 올바른 Amazon SNS 주제를 사용하고 있는지 확인합니다.
  - IAM 서비스 역할을 사용하여 Amazon Rekognition Video에 단일 Amazon SNS 주제에 대한 액세스를 제공한 경우 올바른 Amazon SNS 주제에 대한 권한을 부여했는지 확인합니다. 자세한 내용은 기존 Amazon SNS 주제에 대한 액세스 권한 부여 단원을 참조하십시오.

  - IAM 서비스 역할을 사용하여 Amazon Rekognition Video에 여러 SNS 주제에 대한 액세스를 제공한 경우 올바른 주제를 사용 중이고, 주제의 이름 앞에 AmazonRekognition이 붙어 있는지 확인합니다. 자세한 내용은 여러 Amazon SNS 주제에 대한 액세스 권한 부여 단원을 참조하십시오.
  - AWS Lambda 함수를 사용하는 경우 Lambda 함수가 올바른 Amazon SNS 주제를 구독하는지 확인합니다. 자세한 내용은 Lambda 함수로 팬아웃을 참조하세요.
- Amazon SNS 주제에 대해 Amazon SQS 대기열을 구독한 경우 Amazon SNS 주제에 Amazon SQS 대기열로 메시지를 전송할 권한이 있는지 확인합니다. 자세한 내용은 Amazon SNS 주제에 Amazon SQS 대기열로 메시지를 전송할 권한 부여를 참조하세요.

Amazon SNS 주제의 문제를 해결하는 데 추가 도움이 필요합니다

Amazon SNS와 AWS X-Ray 함께를 사용하여 애플리케이션을 통해 이동하는 메시지를 추적하고 분석할 수 있습니다. 자세한 내용은 Amazon SNS 및 AWS X-Ray 섹션을 참조하세요.

도움이 더 필요한 경우 Amazon Rekognition 포럼 포럼에 질문을 올리거나 AWS 기술 지원 등록을 고려해 보실 수 있습니다.

스트리밍 비디오 이벤트 작업

Amazon Rekognition Video를 사용하면 스트리밍 비디오에서 얼굴을 감지하고 인식하거나 객체를 감지할 수 있습니다. Amazon Rekognition Video는 Amazon Kinesis Video Streams를 사용하여 비디오 스트림을 수신하고 처리합니다. 스트림 프로세서가 비디오 스트림에서 감지할 항목을 표시하는 파라미터를 사용하여 스트림 프로세서를 생성합니다. Rekognition은 스트리밍 비디오 이벤트의 레이블 감지 결과를 Amazon SNS 및 Amazon S3 알림으로 전송합니다. Rekognition은 얼굴 검색 결과를 Kinesis 데이터 스트림에 출력합니다.

얼굴 검색 스트림 프로세서는 컬렉션에서 얼굴을 검색하는 데 `FaceSearchSettings`를 사용합니다. 스트리밍 비디오에서 얼굴을 분석하기 위해 얼굴 검색 스트림 프로세서를 구현하는 방법에 대한 자세한 내용은 the section called “스트리밍 비디오에서 컬렉션의 얼굴 검색” 섹션을 참조하세요.

레이블 감지 스트림 프로세서는 스트리밍 비디오 이벤트에서 사람, 패키지, 반려동물을 검색하는 데 `ConnectedHomeSettings`를 사용합니다. 레이블 감지 스트림 프로세서의 구현 방법에 대한 자세한 내용은 the section called “스트리밍 비디오 이벤트의 레이블 감지” 섹션을 참조하세요.

Amazon Rekognition Video 스트림 프로세서 작업 개요

Amazon Rekognition Video 스트림 프로세서를 시작하고 Amazon Rekognition Video로 비디오를 스트리밍하여 스트리밍 비디오 분석을 시작합니다. Amazon Rekognition Video 스트림 프로세서

를 사용하면 스트림 프로세서를 시작, 중지, 관리할 수 있습니다. 스트림 프로세서를 생성하려면 CreateStreamProcessor를 직접 호출합니다. 얼굴 검색 스트림 프로세서를 생성하기 위한 요청 파라미터에는 Kinesis 비디오 스트림의 Amazon 리소스 이름(ARN), Kinesis 데이터 스트림, 스트리밍 비디오에서 얼굴을 인식하는 데 사용되는 컬렉션의 식별자가 포함됩니다. 보안 모니터링 스트림 프로세서를 생성하기 위한 요청 파라미터에는 Kinesis 비디오 스트림 및 Amazon SNS 주제에 대한 Amazon 리소스 이름(ARN), 비디오 스트림에서 탐지하려는 객체 유형, 출력 결과를 위한 Amazon S3 버킷 정보가 포함됩니다. 또한 스트림 프로세서에 지정한 이름도 포함됩니다.

StartStreamProcessor 작업을 직접 호출하여 비디오 처리를 시작합니다. 스트림 프로세서의 상태 정보를 얻으려면 DescribeStreamProcessor를 직접 호출합니다. 직접 호출할 수 있는 다른 작업으로는 스트림 프로세서에 태그를 지정하기 위한 TagResource, 스트림 프로세서를 삭제하기 위한 DeleteStreamProcessor 등이 있습니다. 얼굴 검색 스트림 프로세서를 사용하는 경우 StopStreamProcessor를 사용하여 스트림 프로세서를 중지할 수도 있습니다. 계정의 스트림 프로세서 목록을 확인하려면 ListStreamProcessors를 직접 호출합니다.

스트림 프로세서가 실행되기 시작한 후, `CreateStreamProcessor`에서 지정한 Kinesis 비디오 스트림을 통해 비디오를 Amazon Rekognition Video로 스트리밍합니다. Kinesis Video Streams SDK의 PutMedia 작업을 사용하여 Kinesis 비디오 스트림으로 비디오를 전송할 수 있습니다. 예제를 보려면 PutMedia API 예제를 참조하십시오.

애플리케이션이 얼굴 검색 스트림 프로세서에서 Amazon Rekognition Video 분석 결과를 사용하는 방법에 대한 자세한 내용은 스트리밍 비디오 분석 결과 읽기 섹션을 참조하세요.

새 스트림 프로세서에 태그 추가

태그를 사용하여 Amazon Rekognition 스트림 프로세서를 식별, 구성, 검색 및 필터링할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다.

`CreateStreamProcessor` 작업을 사용하여 스트림 프로세서를 생성할 때 태그를 추가할 수 있습니다. `Tags` 배열 입력 파라미터에 하나 이상의 태그를 지정합니다. 다음은 태그가 추가된 `CreateStreamProcessor` 요청에 대한 JSON 예제입니다.

```
{
	"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"
        }
}
```

기존 스트림 프로세서에 태그 추가

태그를 사용하여 Amazon Rekognition 스트림 프로세서를 식별, 구성, 검색 및 필터링할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다.

기존 스트림 프로세서에 하나 이상의 태그를 추가하려면 `TagResource` 작업을 사용합니다. 스트림 프로세서의 Amazon 리소스 이름(ARN)(`ResourceArn`)과 추가할 태그(`Tags`)를 지정합니다. 다음 예제는 태그 2개를 추가하는 방법을 보여줍니다.

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

> **Note**
>
> 스트림 프로세서의 Amazon 리소스 이름을 모르는 경우 `DescribeStreamProcessor` 작업을 사용할 수 있습니다.

스트림 프로세서의 태그 나열

태그를 사용하여 Amazon Rekognition 스트림 프로세서를 식별, 구성, 검색 및 필터링할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다.

스트림 프로세서에 붙은 태그를 나열하려면 ListTagsForResource 작업을 사용하고 스트림 프로세서의 ARN(ResourceArn)을 지정합니다. 응답은 지정된 스트림 프로세서에 연결된 태그 키 및 값의 맵입니다.

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

출력된 결과에 스트림 프로세서에 연결된 태그 목록이 표시됩니다.

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

스트림 프로세서에서 태그 삭제

태그를 사용하여 Amazon Rekognition 스트림 프로세서를 식별, 구성, 검색 및 필터링할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다.

스트림 프로세서에서 하나 이상의 태그를 삭제하려면 UntagResource 작업을 사용합니다. 제거하려는 모델의 ARN(ResourceArn)과 태그 키(Tag-Keys)를 지정합니다.

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

또는 다음 형식으로 tag-key를 지정할 수도 있습니다.

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

오류 처리

이 단원에서는 런타임 오류와 그러한 오류를 처리하는 방법을 설명합니다. 또한 Amazon Rekognition 관련 오류 메시지 및 코드를 설명합니다.

주제

- 오류 구성 요소
- 오류 메시지 및 코드
- 애플리케이션에서의 오류 처리

오류 구성 요소

프로그램이 요청을 보내면 Amazon Rekognition은 요청 처리를 시도합니다. 요청이 성공하면 Amazon Rekognition은 요청된 작업의 결과와 함께 HTTP 성공 상태 코드(`200 OK`)를 반환합니다.

요청이 실패하면 Amazon Rekognition은 오류를 반환합니다. 오류에는 세 가지 구성 요소가 있습니다.

- HTTP 상태 코드(예: `400`).
- 예외 이름(예: `InvalidS3ObjectException`).
- 오류 메시지(예: `Unable to get object metadata from S3. Check object key, region and/or access permissions.`).

AWS SDK는 적절한 조치를 취할 수 있도록 애플리케이션에 오류를 전파합니다. 예를 들어 Java 프로그램에서는 `ResourceNotFoundException`을 처리하기 위해 `try-catch` 로직을 작성할 수 있습니다.

AWS SDK를 사용하지 않는 경우, Amazon Rekognition에서 하위 수준 응답의 내용을 구문 분석해야 합니다. 다음은 그러한 응답의 예입니다.

```
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."}
```

오류 메시지 및 코드

다음은 Amazon Rekognition이 반환하는 예외 목록을 HTTP 상태 코드를 기준으로 그룹화한 목록입니다. 재시도 가능?이 예라면 동일한 요청을 다시 제출할 수 있습니다. 재시도 가능?이 아니요라면 새로운 요청을 제출하기 전에 클라이언트 측에서 문제를 해결해야 합니다.

HTTP 상태 코드 400

HTTP `400` 상태 코드는 요청에 문제가 있음을 나타냅니다. 몇 가지 문제의 예로는 인증 실패, 필수 파라미터 누락 또는 작업의 프로비저닝된 처리량 초과 등입니다. 요청을 다시 제출하기 전에 애플리케이션의 문제를 해결해야 합니다.

AccessDeniedException

메시지: <Operation> 작업을 호출할 때 오류가 발생했습니다(AccessDeniedException). 사용자(<User ARN>)는 리소스(<Resource ARN>)에서 <Operation> 작업을 수행할 권한이 없습니다.

작업을 수행할 권한이 없습니다. 권한 있는 사용자 또는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용하여 작업을 수행하십시오.

재시도 가능? 아니요

GroupFacesInProgressException

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

기존 작업이 완료된 후에 작업을 재시도해 보십시오.

재시도 가능? 아니요

IdempotentParameterMismatchException

메시지: 제공된 ClientRequestToken(<Token>)은 이미 사용 중입니다.

ClientRequestToken 입력 파라미터가 작업에서 재사용되었지만 하나 이상의 다른 입력 파라미터가 이전의 작업 호출과 다릅니다.

재시도 가능? 아니요

ImageTooLargeException

메시지: 이미지가 너무 큽니다.

입력 이미지의 크기가 허용된 한도를 초과합니다. DetectProtectiveEquipment를 직접 호출하는 경우 이미지 크기 또는 해상도가 허용된 한도를 초과합니다. 자세한 내용은 Amazon Rekognition의 가이드라인 및 할당량 단원을 참조하십시오.

재시도 가능? 아니요

InvalidImageFormatException

메시지: 요청에 잘못된 이미지 형식이 있습니다.

제공된 이미지 형식은 지원되지 않습니다. 지원되는 이미지 형식(.JPEG 및 .PNG)을 사용하십시오. 자세한 내용은 Amazon Rekognition의 가이드라인 및 할당량 단원을 참조하십시오.

재시도 가능? 아니요

InvalidPaginationTokenException

메시지

- 잘못된 토큰
- 잘못된 매김 토큰

요청의 매김 토큰이 잘못되었습니다. 토큰이 만료되었을 수 있습니다.

재시도 가능? 아니요

InvalidParameterException

메시지: 요청에 잘못된 파라미터가 있습니다.

입력 파라미터가 제약 조건을 위반했습니다. 파라미터를 확인한 다음 API 작업을 다시 호출하십시오.

재시도 가능? 아니요

InvalidS3ObjectException

메시지:

- 요청에 잘못된 S3 객체가 있습니다.
- Unable to get object metadata from S3. Check object key, region and/or access permissions.

Amazon Rekognition이 요청에 지정된 S3 객체에 액세스할 수 없습니다. 자세한 내용은 S3 액세스 구성: AWS S3 액세스 권한 관리 단원을 참조하십시오. 문제 해결 정보는 Amazon S3 문제 해결을 참조하십시오.

재시도 가능? 아니요

LimitExceededException

메시지:

- 스트림 프로세서의 제한이 계정의 한도(<Current Limit>)를 초과했습니다.
- 사용자(<User ARN>)를 위한 열린 작업 <Number of Open Jobs>개. 최대 한도: <Maximum Limit>

Amazon Rekognition 서비스 제한을 초과했습니다. 예를 들어 너무 많은 Amazon Rekognition Video 작업을 동시에 시작하면서 `StartLabelDetection` 등의 작업을 직접 호출할 경우에는 동시 실행 작업의 수가 Amazon Rekognition 서비스 제한 미만이 될 때까지 `LimitExceededException` 예외(HTTP 상태 코드: 400)가 발생합니다.

재시도 가능? 아니요

ProvisionedThroughputExceededException

메시지:

- 프로비저닝된 속도가 초과되었습니다.
- S3 다운로드 한도가 초과되었습니다.

요청의 수가 처리량 한도를 초과했습니다. 자세한 내용은 Amazon Rekognition 서비스 한도를 참조하세요.

한도 증가를 요청하려면 the section called “TPS 할당량 변경을 위한 사례 생성”의 지침을 따르세요.

재시도 가능? 예

ResourceAlreadyExistsException

메시지: 모음의 ID(<Collection Id>)가 이미 존재합니다.

지정된 ID의 모음이 이미 존재합니다.

재시도 가능? 아니요

ResourceInUseException

메시지:

- 스트림 프로세서의 이름이 이미 사용 중입니다.
- 지정된 리소스가 이미 사용 중입니다.
- 프로세서를 스트림 중지에 사용할 수 없습니다.
- 스트림 프로세서를 삭제할 수 없습니다.

리소스를 사용할 수 있을 때 다시 시도하십시오.

재시도 가능? 아니요

ResourceNotFoundException

메시지: API 호출에 따라 메시지가 달라집니다.

지정된 리소스가 존재하지 않습니다.

재시도 가능? 아니요

ThrottlingException

메시지: 속도를 줄이십시오. 요청 속도가 급증했습니다.

요청 증가 속도가 너무 빠릅니다. 요청 속도를 줄이고 조금씩 늘리십시오. 백오프를 지수적으로 수행하고 다시 시도하는 것이 좋습니다. 기본적으로 AWS SDKs 자동 재시도 로직과 지수 백오프를 사용합니다. 자세한 내용은 AWS에서의 오류 재시도 및 지수 백오프 단원과 지수 백오프 및 Jitter 단원을 참조하십시오.

재시도 가능? 예

VideoTooLargeException

메시지: 비디오 크기 <Video Size>(바이트)가 최대 한도인 <Max Size>바이트를 초과합니다.

제공된 미디어의 파일 크기 또는 재생 시간이 너무 크거나 깁니다. 자세한 내용은 Amazon Rekognition의 가이드라인 및 할당량 단원을 참조하십시오.

재시도 가능? 아니요

HTTP 상태 코드 5xx

HTTP 5xx 상태 코드는 AWS가 해결해야 하는 문제를 나타냅니다. 일시적 오류일 수 있습니다. 그 경우, 성공할 때까지 요청을 재시도할 수 있습니다. 그렇지 않은 경우 AWS Service Health Dashboard에서 서비스 운영 문제가 있는지 확인하십시오.

InternalServerError(HTTP 500)

메시지: 내부 서버 오류가 발생했습니다.

Amazon Rekognition에 서비스 문제가 발생했습니다. 호출을 다시 시도하세요. 백오프를 지수적으로 수행하고 다시 시도해야 합니다. 기본적으로 AWS SDK는 자동 재시도 로직과 지수 백오프를 사용합니다. 자세한 내용은 AWS에서의 오류 재시도 및 지수 백오프 단원과 지수 백오프 및 Jitter 단원을 참조하십시오.

재시도 가능? 예

ThrottlingException(HTTP 500)

메시지: 서비스를 사용할 수 없습니다.

Amazon Rekognition이 요청을 일시적으로 처리할 수 없습니다. 호출을 다시 시도하세요. 백오프를 지수적으로 수행하고 다시 시도하는 것이 좋습니다. 기본적으로 AWS SDKs 자동 재시도 로직과 지수 백오프를 사용합니다. 자세한 내용은 AWS에서의 오류 재시도 및 지수 백오프 단원과 지수 백오프 및 Jitter 단원을 참조하십시오.

재시도 가능? 예

애플리케이션에서의 오류 처리

애플리케이션의 원활한 실행을 위해서는 오류를 발견하여 대응할 수 있는 로직을 추가해야 합니다. 일반적인 접근법에는 `try-catch` 블록 또는 `if-then` 문의 사용이 포함됩니다.

AWS SDK는 자체적으로 재시도 및 오류 검사를 실행합니다. AWS SDK 사용 중에 오류가 발생하면 해당 오류 코드와 설명이 문제 해결에 도움이 될 수 있습니다.

또한 응답에서 `Request ID`도 확인해야 합니다. 문제 진단을 위해 AWS Support와 협력해야 하는 경우 `Request ID`가 도움이 될 수 있습니다.

다음의 Java 코드 조각은 이미지의 객체 감지를 시도하고, 기초적인 오류 처리를 수행합니다. (이 경우, 사용자에게 요청이 실패했음을 알립니다.)

```
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());
}
```

이 코드 조각에서 `try-catch` 구문은 두 가지 상이한 예외를 처리합니다.

- `AmazonRekognitionException` - 이 예외는 클라이언트 요청이 Amazon Rekognition에 올바르게 전송되었지만, Amazon Rekognition이 요청을 처리할 수 없고 대신 오류 응답을 반환하는 경우에 발생합니다.
- `AmazonClientException` - 이 예외는 클라이언트가 서비스로부터 응답을 받을 수 없거나 서비스의 응답을 구문 분석할 수 없는 경우에 발생합니다.

FedRAMP 인증 서비스로 Amazon Rekognition 사용

AWS FedRAMP 규정 준수 프로그램에는 Amazon Rekognition이 FedRAMP 인증 서비스로 포함되어 있습니다. 연방 또는 상업 고객인 경우 서비스를 사용하여 중요 워크로드를 중간 영향 수준 이하의 데이터로 AWS 미국 동부 및 미국 서부에서 처리하고 저장할 수 있습니다. 서비스를 사용하여 중요 워크로드를 AWS GovCloud(미국) 지역의 승인 범위를 높은 영향 수준 이하의 데이터로 처리할 수 있습니다. FedRAMP 규정 준수에 대한 자세한 내용은 AWS FedRAMP 규정 준수를 참조하십시오.

FedRAMP 규정을 준수하려면 FIPS(Federal Information Processing Standard) 엔드포인트를 사용할 수 있습니다. 이를 통해 중요한 정보로 작업할 때 FIPS 140-2의 검증된 암호화 모듈에 액세스할 수 있습니다. FIPS 엔드포인트에 대한 자세한 내용은 FIPS 140-2 개요를 참조하십시오.

AWS Command Line Interface(AWS CLI) 또는 AWS SDK 중 하나를 사용하여 Amazon Rekognition에서 사용하는 엔드포인트를 지정할 수 있습니다.

Amazon Rekognition에서 사용할 수 있는 엔드포인트는 Amazon Rekognition 리전 및 엔드포인트를 참조하십시오.

다음은 Amazon Rekognition 개발자 안내서의 컬렉션 나열 토픽에서 가져온 예시입니다. Amazon Rekognition에 액세스할 수 있는 리전 및 FIPS 엔드포인트를 보여주도록 수정되었습니다.

Java

Java의 경우 Amazon Rekognition 클라이언트를 구성할 때 `withEndpointConfiguration` 메서드를 사용합니다. 다음 예에서는 미국 동부(버지니아 북부) 지역에서 FIPS 엔드포인트를 사용하는 컬렉션을 보여줍니다.

```
//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

AWS CLI의 경우 `--endpoint-url` 인수를 사용하여 Amazon Rekognition에 액세스할 수 있는 엔드포인트를 지정합니다. 다음 예에서는 미국 동부(오하이오) 지역에서 FIPS 엔드포인트를 사용하는 컬렉션을 보여줍니다.

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

Python

Python의 경우 boto3.client 함수에서 `endpoint_url` 인수를 사용합니다. 지정하려는 엔드포인트로 설정합니다. 다음 예에서는 미국 서부(오레곤) 지역에서 FIPS 엔드포인트를 사용하는 컬렉션을 보여줍니다.

```
#Copyright 2018 Amazon.com, Inc. or its 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()
```

센서, 입력 이미지 및 비디오 모범 사례

이 섹션에는 Amazon Rekognition 사용에 대한 모범 사례 정보가 포함되어 있습니다. 이러한 모범 사례는 호출하는 작업에서 최적의 성능을 얻는 데 도움이 됩니다. 작업에서 예상 결과를 얻는 데 어려움을 겪고 있는 경우 여기에 설명된 모범 사례를 따르는 것이 좋습니다.

이미지 작업의 지연 시간에 대한 자세한 내용은 다음을 참조하세요.

- the section called “Amazon Rekognition Image 작업 지연 시간”

얼굴 비교 및 얼굴 검색 작업을 수행하려면 특정 모범 사례를 따라 이미지에서 얼굴을 찾아야 합니다. 이러한 요구 사항은 다음 링크에도 설명되어 있습니다.

- the section called “얼굴 비교 입력 이미지에 대한 권장 사항”
- the section called “컬렉션에서 얼굴을 검색할 때의 권장 사항”

다음 섹션에서는 Amazon Rekognition에서 분석할 수 있는 각 미디어 유형에 대해 카메라를 설정하는 방법을 다룹니다.

- the section called “카메라 설정 권장 사항(이미지 및 비디오)”
- the section called “카메라 설정 권장 사항(저장 및 스트리밍 비디오)”
- the section called “카메라 설정 권장 사항(스트리밍 비디오)”

또한 Face Liveness 작업에는 Liveness 검사 도구에서 최상의 성능을 얻기 위해 따라야 하는 자체 모범 사례도 있습니다.

- the section called “Face Liveness 사용에 대한 권장 사항”

Amazon Rekognition Image 작업 지연 시간

Amazon Rekognition Image 작업의 지연 시간을 최소화하려면 다음을 고려하세요.

- 이미지가 포함된 Amazon S3 버킷의 리전과 Amazon Rekognition Image API 작업에 사용하는 리전이 일치해야 합니다.
- 이미지 바이트로 Amazon Rekognition Image 작업을 직접 호출하는 것이 이미지를 Amazon S3 버킷에 업로드한 다음 Amazon Rekognition Image 작업에서 업로드된 이미지를 참조하는 것보다 빠릅니

다. 실시간에 가까운 처리를 위해 이미지를 Amazon Rekognition Image에 업로드하는 경우 이 접근 방식을 고려해 보세요. 예를 들어 IP 카메라에서 업로드한 이미지 또는 웹 포털을 통해 업로드한 이미지입니다.

- 이미지가 Amazon S3 버킷에 이미 있는 경우, Amazon Rekognition Image 작업에서 해당 이미지를 참조하는 것이 이미지 바이트를 일일이 작업에 전달하는 것보다 더 빠를 것입니다.

얼굴 비교 입력 이미지에 대한 권장 사항

얼굴 비교 작업에 사용되는 모델은 다양한 포즈, 얼굴 표정, 연령 범위, 회전, 조명 조건 및 크기로 작동하도록 설계되었습니다. CompareFaces용 레퍼런스 사진을 선택할 때나 IndexFaces를 사용하여 얼굴을 컬렉션에 추가할 때는 다음 지침을 따르는 것이 좋습니다.

얼굴 작업을 위한 입력 이미지에 대한 일반 권장 사항

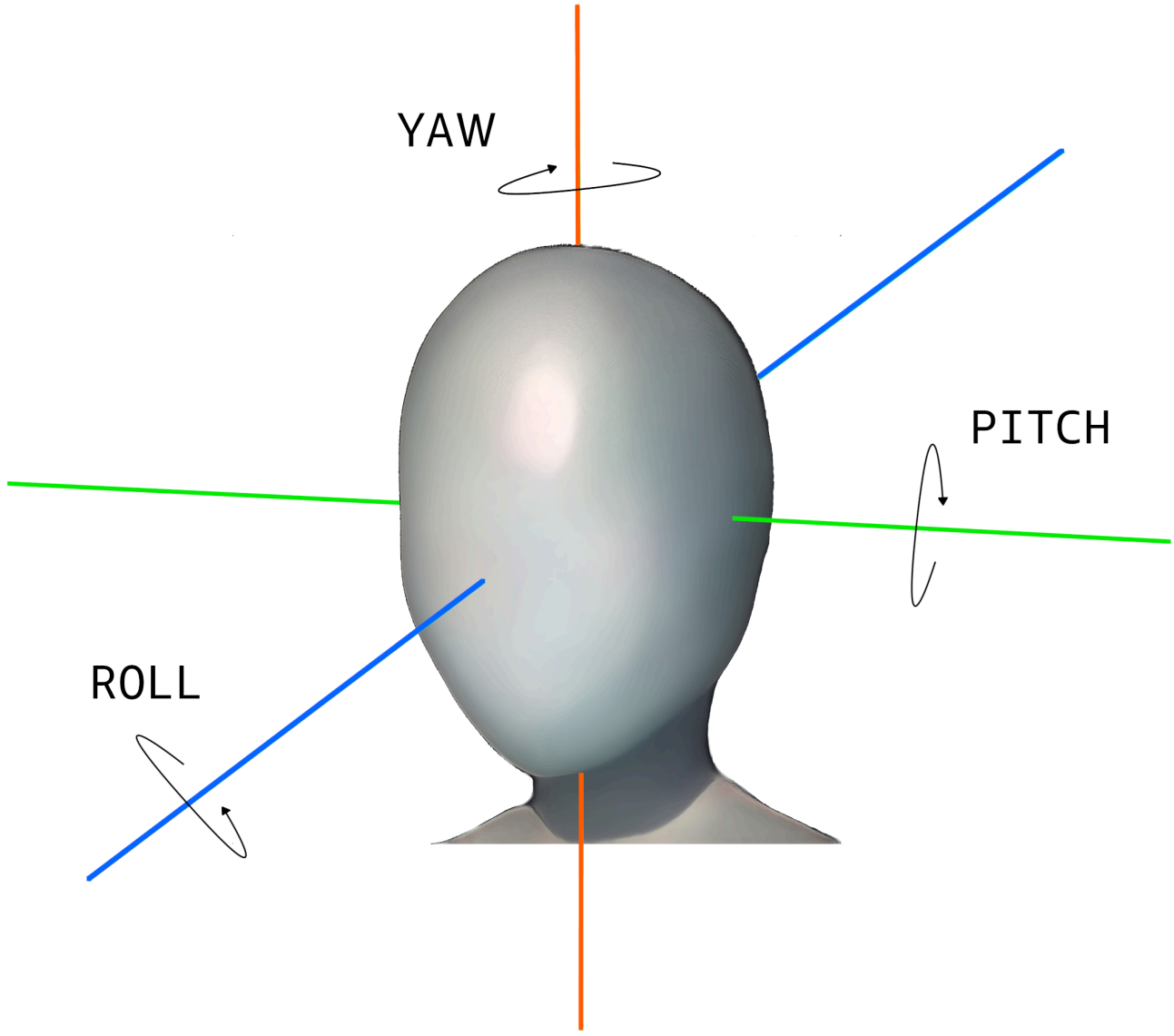
- 밝고 선명한 이미지를 사용하세요. 피사체와 카메라 움직임 때문에 흐려질 수 있는 이미지는 가능한 한 사용하지 않아야 합니다. DetectFaces를 사용하여 얼굴의 밝기와 선명도를 파악할 수 있습니다.
- 시선 감지를 위해 원본 이미지를 원본 크기 및 품질로 업로드하는 것을 권고합니다.
- 얼굴이 권장 각도 범위 이내인 이미지를 사용합니다. 피치는 아래로 30도 미만이고 위로 45도 미만이어야 합니다. 요는 양쪽 방향으로 45도 미만이어야 합니다. 롤에는 제한이 없습니다.
- 두 눈이 잘 보이는 얼굴 이미지를 사용합니다.
- 가려진 부분이 없고 화면을 꽉 채우지 않은 상태의 얼굴 이미지를 사용합니다. 이미지에 사람의 머리와 어깨가 포함되어 있어야 합니다. 얼굴 경계 상자로 자른 상태여서는 안 됩니다.
- 머리띠, 마스크 등 얼굴을 가리는 아이템을 피합니다.
- 얼굴이 이미지의 많은 부분을 차지하는 이미지를 사용합니다. 얼굴이 이미지의 더 많은 부분을 차지하는 이미지가 보다 높은 정확도로 일치됩니다.
- 이미지 해상도가 충분히 커야 합니다. Amazon Rekognition은 최대 1920 x 1080의 이미지 해상도에서 50 x 50픽셀의 작은 얼굴까지 인식할 수 있습니다. 이미지 해상도가 높을수록 최소 얼굴 크기를 크게 지정해야 합니다. 최소 크기보다 큰 얼굴은 보다 정확한 얼굴 비교 결과 세트를 제공합니다.
- 색상 이미지를 사용합니다.
- 그림자 같이 불균일한 조명이 아니라 얼굴에 균일한 조명을 비춘 이미지를 사용합니다.
- 배경과 충분한 대비를 보이는 이미지를 사용하세요. 고대비 흑백 배경이 효과적입니다.
- 높은 정확도가 요구되는 애플리케이션에서는 입을 다물고 미소를 전혀 또는 거의 짓지 않은 중립적인 표정의 얼굴 이미지를 사용합니다.

컬렉션에서 얼굴을 검색할 때의 권장 사항

- 컬렉션에서 얼굴을 검색할 때는 최근 얼굴 이미지가 인덱싱되어 있는지 확인하세요.
- IndexFaces를 사용하여 모음을 생성할 때 특정 개인에 대해 피치 및 요(권장 각도 이내)가 다른 여러 개의 이미지를 사용합니다. 정면 이미지, 얼굴을 왼쪽으로 돌린 이미지(요 45도 이내), 얼굴을 오른쪽으로 돌린 이미지(요 45도 이내), 얼굴을 아래로 기울인 이미지(피치 30도 이내), 얼굴을 위로 젖힌 이미지(피치 45도 이내) 등 개인당 최소 5개의 이미지를 인덱싱하는 것이 좋습니다. 같은 사람에 속하는 얼굴 인스턴스를 추적하려면, 인덱싱되는 이미지에 얼굴이 하나뿐인 경우 외부 이미지 ID 속성을 사용할 것을 고려하십시오. 예를 들어 John_Doe_1.jpg, … John_Doe_5.jpg와 같은 외부 이미지 ID를 포함한 John Doe의 이미지 5개를 컬렉션 내에서 추적할 수 있습니다.

카메라 설정 권장 사항(이미지 및 비디오)

다음은 얼굴 비교 입력 이미지에 대한 권장 사항에 추가되는 권장 사항입니다.



- 이미지 해상도 – 이미지 해상도에는 최소 요구 사항이 없지만, 최대 1920 x 1080 픽셀의 이미지 해상도에서 얼굴이 50 x 50 픽셀 이상이어야 합니다. 이미지 해상도가 높을수록 최소 얼굴 크기를 크게 지정해야 합니다.

> **Note**
>
> 위 권장 사항은 카메라의 기반 해상도를 기준으로 합니다. 저해상도 이미지에서 고해상도 이미지를 생성하면 얼굴 검색에 필요한 결과를 얻을 수 없습니다(이미지 업샘플링 과정에서 생성되는 아티팩트 때문).

- 카메라 각도 – 카메라 각도에는 피치, 롤, 요의 세 가지 측정값이 있습니다.
  - 피치 – 카메라가 아래를 향할 경우 30도 미만, 위를 향할 때는 45도 미만의 피치를 권장합니다.
  - 롤 – 해당 항목에는 최소 요구 사항이 없습니다. Amazon Rekognition은 어떤 양의 롤이라도 처리할 수 있습니다.
  - 요 – 각 방향에서 45도 미만의 요를 권장합니다.

  카메라가 포착하는 축에서 얼굴 각도는 현장을 향하는 카메라 각도와 현장에서 피사 인물의 머리가 향하는 방향의 조합입니다. 예를 들어 카메라가 30도 아래를 향하고 있고 사람이 머리를 30도 숙이고 있을 경우 카메라를 통해 포착되는 실제 얼굴 피치는 60도입니다. 이 경우에는 Amazon Rekognition이 얼굴을 인식할 수 없습니다. 일반적으로 사람들이 30도 이하의 전체 피치(얼굴과 카메라의 조합)로 카메라를 바라본다는 가정 하에 카메라 각도를 설정하는 것이 좋습니다.
- 카메라 줌 – 최소 얼굴 해상도 권장 사항인 50x50 픽셀에 따라 이 카메라 설정을 조정해야 합니다. 원하는 얼굴이 50x50 픽셀 이상의 해상도가 되도록 카메라 줌 설정을 사용하는 것이 좋습니다.
- 카메라 높이 – 카메라 피치 권장 사항에 따라 이 파라미터를 조정해야 합니다.

카메라 설정 권장 사항(저장 및 스트리밍 비디오)

다음은 카메라 설정 권장 사항(이미지 및 비디오)에 추가되는 권장 사항입니다.

- 코덱은 h.264를 사용해야 합니다.
- 권장 프레임 속도는 30fps입니다. (5fps보다 작아서는 안 됩니다.)
- 권장 인코더 비트레이트는 3Mbps입니다. (1.5Mbps보다 작아서는 안 됩니다.)
- 프레임 속도 대 해상도 – 인코더 비트레이트가 제약 조건일 경우, 더 나은 얼굴 검색 결과를 얻기 위해 프레임 속도보다는 프레임 해상도를 높이는 것이 좋습니다. 그러면 할당된 비트레이트 내에서 Amazon Rekognition 사용을 통해 가장 좋은 품질의 프레임을 얻을 수 있습니다. 그러나 단점이 있습니다. 낮은 프레임 속도로 인해 카메라가 빠른 움직임을 놓치게 됩니다. 주어진 설정에서 이들 두 파라미터의 상반 관계를 이해하는 것이 중요합니다. 예를 들어 최대 가능 비트레이트가 1.5Mbps일 경우, 카메라는 1080p @ 5fps 또는 720p @ 15fps를 캡쳐할 수 있습니다. 권장 얼굴 해상도인 50x50 픽셀이 충족되는 한, 둘 사이에서의 선택은 애플리케이션에 따라 달라집니다.

카메라 설정 권장 사항(스트리밍 비디오)

다음은 카메라 설정 권장 사항(저장 및 스트리밍 비디오)에 추가되는 권장 사항입니다.

스트리밍 애플리케이션에서 추가 제약 사항은 인터넷 대역폭입니다. 라이브 비디오의 경우, Amazon Rekognition은 Amazon Kinesis Video Streams만을 입력으로 받아들입니다. 인코더 비트레이트와 가용 네트워크 대역폭 사이의 의존 관계를 이해해야 합니다. 적어도 가용 대역폭이 카메라가 라이브 스트림을 인코딩하는 데 사용하는 비트레이트를 지원해야 합니다. 그러면 카메라가 무엇을 캡처하든 Amazon Kinesis Video Streams를 통해 재생될 수 있습니다. 가용 대역폭이 인코더 비트레이트보다 낮은 경우 Amazon Kinesis Video Streams가 네트워크 대역폭을 기준으로 비트를 드롭합니다. 그러면 비디오 품질이 낮아집니다.

일반적인 스트리밍 설정에서는 여러 대의 카메라가 스트림을 중계하는 네트워크 허브에 연결됩니다. 이 경우 허브에 연결된 모든 카메라에서 전송되는 스트림의 누적 합계를 네트워크 대역폭이 수용해야 합니다. 예를 들어 허브에 1.5Mbps로 인코딩하는 카메라 5대가 연결된 경우, 가용 네트워크 대역폭은 최소 7.5Mbps여야 합니다. 패킷 드롭을 방지하려면, 카메라와 허브 사이의 드롭된 연결로 인한 지터를 수용하도록 네트워크 대역폭을 7.5Mbps 이상으로 유지할 것을 고려해야 합니다. 실제 값은 내부 네트워크의 안정성에 따라 달라질 수 있습니다.

Face Liveness 사용에 대한 권장 사항

Rekognition Face Liveness를 사용할 때는 다음 모범 사례를 따르는 것이 좋습니다.

- 사용자는 너무 어둡거나 너무 밝지 않고 조명이 대체로 균일한 환경에서 Face Liveness 검사를 완료해야 합니다.
- 사용자는 웹 브라우저에서 검사를 실행할 때 디스플레이 화면 밝기를 최대 수준으로 높여야 합니다. 모바일 네이티브 SDK는 디스플레이 밝기를 자동으로 조정합니다.
- 사용 사례의 특성을 반영하는 신뢰도 점수 임계값을 선택하세요. 보안에 대한 우려가 더 큰 사용 사례의 경우 높은 임계값을 사용하세요.
- 감사 이미지에 대해 정기적으로 인적 검토 검사를 실행하여 설정한 신뢰 임계값에서 스푸핑 공격이 경감되는지 확인하세요.
- 사진에 민감하거나 Rekognition을 사용한 얼굴 인증을 원하지 않는 사용자에게는 얼굴 인증을 할 수 있는 다른 대안을 제공하세요.
- 사용자 애플리케이션에 얼굴 인증 검사 점수를 보내거나 표시하지 마세요. 통과 또는 실패 신호만 보내세요.

- 단일 디바이스에서 인증 검사 실패는 3분에 5회만 허용합니다. 5회 실패한 경우 30~60분 동안 사용자를 타임아웃시키세요. 같은 패턴이 3~5번 반복해서 나타나는 경우 사용자 디바이스에서의 추가 직접 호출을 차단하세요.
- 워크플로에 준비 화면을 구현하여 사용자가 Face Liveness 검사를 더 쉽게 통과할 수 있도록 해 주세요.
- Face Liveness의 콘텐츠 처리, 저장, 사용 및 전송에 대해 최종 사용자에게 법적으로 적절한 개인정보 처리방침을 제공하고 필요한 동의를 받을 할 책임은 귀하에게 있습니다.

객체 및 개념 감지

이 섹션에서는 Amazon Rekognition Image 및 Amazon Rekognition Video를 사용한 이미지와 비디오에서의 레이블 감지에 대한 정보를 다룹니다.

레이블 또는 태그는 콘텐츠를 기반으로 이미지나 비디오에서 발견되는 객체 또는 개념(장면과 행동을 포함한)을 가리킵니다. 예를 들면 열대 해변에 있는 사람들을 찍은 이미지에는 야자수(객체), 해변(장면), 달리기(행동), 야외(개념)와 같은 레이블이 포함될 수 있습니다.

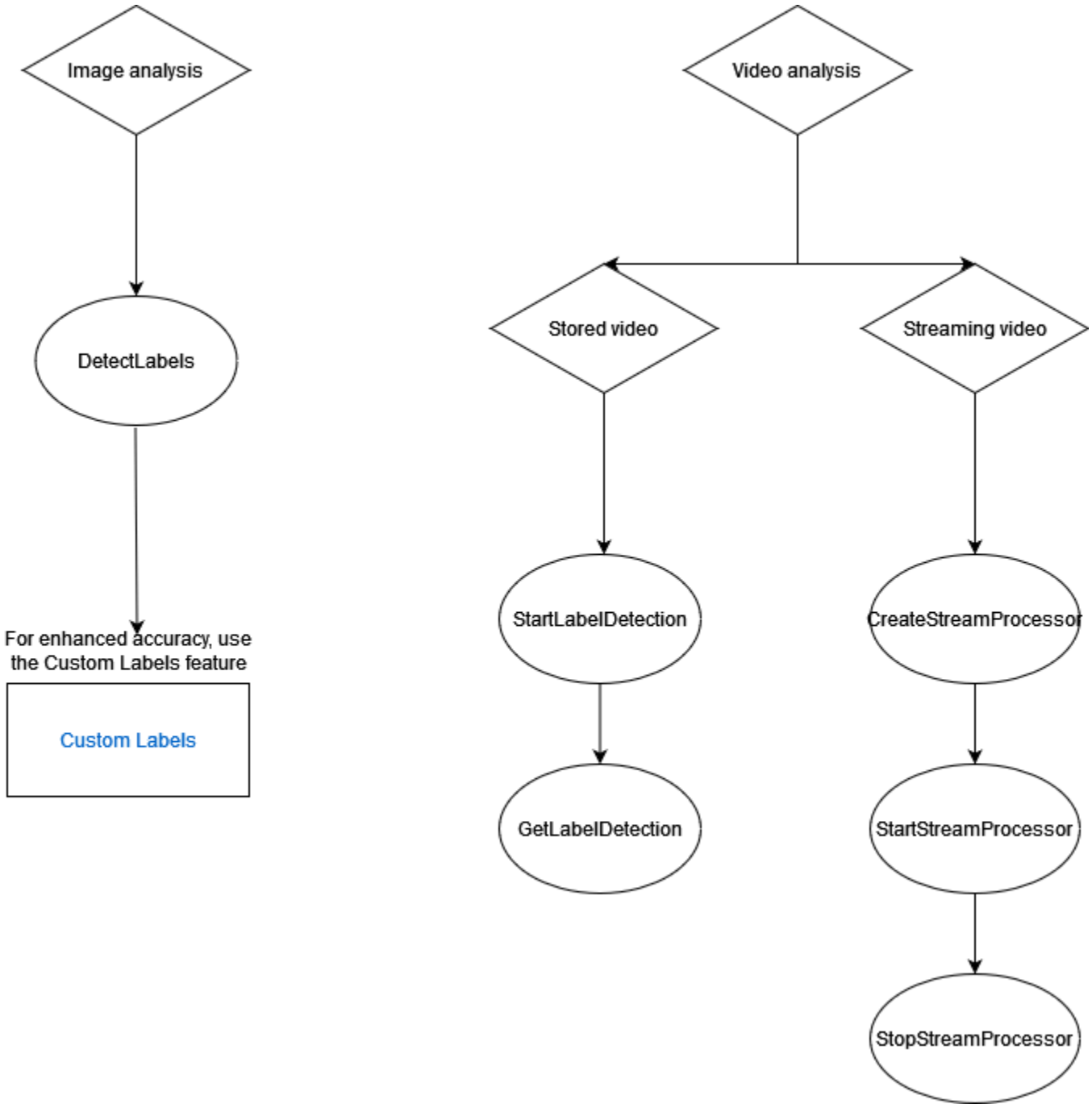
Rekognition 레이블 감지 작업에서 지원하는 레이블

- Amazon Rekognition에서 지원하는 레이블 및 객체 경계 상자의 최신 목록을 다운로드하려면 여기를 클릭하세요.
- 레이블 및 객체 경계 상자의 이전 목록을 다운로드하려면 여기를 클릭하세요.

> **Note**
>
> Amazon Rekognition은 특정 이미지에 나오는 사람의 신체적 외양을 기반으로 성별 이분법적인(남성, 여성, 소녀 등) 예측을 내놓습니다. 이러한 종류의 예측은 사람의 성 정체성을 범주화하도록 설계되지 않았으므로 이러한 결정을 내리는 데 Amazon Rekognition을 사용하지 않아야 합니다. 예를 들어, 연기를 위해 긴 머리 가발과 귀걸이를 착용한 남성 배우를 여성으로 예측할 수 있습니다.
>
> Amazon Rekognition을 사용한 성별 이분법적 예측은 특정 사용자 식별 없이 전체 성별 분포 통계를 분석해야 하는 사용 사례에 가장 적합합니다. 소셜 미디어 플랫폼에서 남성 대비 여성 사용자의 비율을 예로 들 수 있습니다.
>
> 개인의 권리, 사생활 또는 서비스 액세스에 영향을 미칠 수 있는 결정을 내리는 데에는 성별 이분법적 예측을 사용하지 않는 것을 권장합니다.

Amazon Rekognition은 레이블을 영어로 반환합니다. Amazon Translate를 사용하여 영어 레이블을 다른 언어로 번역할 수 있습니다.

다음 다이어그램은 Amazon Rekognition Image 또는 Amazon Rekognition Video 작업 사용 목표에 따른 직접 호출 작업의 순서를 보여줍니다.



레이블 응답 객체

경계 상자

Amazon Rekognition Image 및 Amazon Rekognition Video는 차량, 가구, 의류, 반려동물 등 일반적 객체 레이블에 대해 경계 상자를 반환할 수 있습니다. 경계 상자 정보는 이보다 일반적이지 않은 객체 레이블에 대해서는 반환되지 않습니다. 경계 상자를 사용하여 이미지에서 객체의 정확한 위치를 찾거나,

감지된 객체의 인스턴스 수를 계산하거나, 경계 상자 치수를 사용하여 객체의 크기를 측정할 수 있습니다.

예를 들어 다음 이미지에서 Amazon Rekognition Image는 사람의 존재, 스케이트보드, 주차된 차량 및 기타 정보를 감지할 수 있습니다. Amazon Rekognition Image는 감지된 사람과 자동차나 바퀴와 같은 기타 감지된 객체에 대해서도 경계 상자를 반환합니다.

신뢰도 점수

Amazon Rekognition Video와 Amazon Rekognition Image는 감지된 각 레이블의 정확성에 대해 Amazon Rekognition이 얼마나 확신하는지를 나타내는 백분율 점수를 제공합니다.

상위 개념

Amazon Rekognition Image와 Amazon Rekognition Video는 상위 레이블의 계층적 분류 체계를 사용하여 레이블을 분류합니다. 예를 들어 도로를 횡단하는 사람이 보행자로 감지될 수 있습니다. 보행자의 부모 레이블은 인물입니다. 응답에서 두 레이블이 모두 반환됩니다. 모든 상위 레이블이 반환되고, 지

정된 레이블이 부모 및 기타 상위 레이블의 목록을 포함합니다. 예를 들어, 존재할 경우 조부모 및 증조부모 레이블이 포함됩니다. 부모 레이블을 사용하여 관련된 레이블의 그룹을 만들고 하나 이상의 이미지에서 비슷한 레이블을 쿼리하도록 허용할 수 있습니다. 예를 들어 모든 차량을 쿼리하면 한 이미지에서 자동차가 반환되고 다른 이미지에서 오토바이가 반환될 수 있습니다.

Categories

Amazon Rekognition Image 및 Amazon Rekognition Video는 레이블 카테고리에 대한 정보를 반환합니다. 레이블은 '차량 및 자동차', '식음료'와 같이 공통 기능 및 맥락에 기반하여 개별 레이블을 그룹화하는 카테고리의 일부입니다. 레이블 카테고리는 상위 카테고리의 하위 카테고리일 수 있습니다.

별칭

Amazon Rekognition Image와 Amazon Rekognition Video는 레이블을 반환하는 것 외에도 레이블과 연결된 모든 별칭을 반환합니다. 별칭은 의미가 같은 레이블 또는 반환된 기본 레이블과 시각적으로 서로 바꿀 수 있는 레이블을 말합니다. 예를 들어, 'Cell Phone'은 'Mobile Phone'의 별칭입니다.

이전 버전에서는 Amazon Rekognition Image가 'Mobile Phone'을 포함하는 동일한 기본 레이블 이름 목록에서 'Cell Phone'과 같은 별칭을 반환했습니다. Amazon Rekognition Image는 이제 “별칭” 필드에 ‘Cell Phone’을, 기본 레이블 이름 목록에 'Mobile Phone'을 반환합니다. 애플리케이션이 이전 버전의 Rekognition에서 반환된 구조에 의존한다면 이미지 또는 동영상 레이블 감지 작업에서 반환되는 현재 응답을 모든 레이블과 별칭이 기본 레이블로 반환되는 이전의 응답 구조로 변환해야 할 수 있습니다.

DetectLabels API(이미지의 레이블 감지용)의 현재 응답을 이전 응답 구조로 변환해야 하는 경우 DetectLabels 응답의 변환의 코드 예제를 참조하세요.

GetLabelDetection API(저장된 동영상의 레이블 감지용)의 현재 응답을 이전 응답 구조로 변환해야 하는 경우 GetLabelDetection 응답의 변환의 코드 예제를 참조하세요.

이미지 속성

Amazon Rekognition Image는 전체 이미지의 이미지 품질(선명도, 밝기, 대비)에 대한 정보를 반환합니다. 이미지의 전경과 배경에 대해서도 선명도와 밝기가 반환됩니다. 또한 이미지 속성을 사용하여 전체 이미지, 전경, 배경 및 경계 상자가 있는 객체의 주 색상을 감지할 수도 있습니다.

다음은 현재 이미지에 대한 DetectLabels 작업의 응답에 포함된 ImageProperties 데이터의 예입니다.

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	

Amazon Rekognition Video에서는 이미지 속성을 사용할 수 없습니다.

모델 버전

Amazon Rekognition Image 및 Amazon Rekognition Video는 이미지 또는 저장된 비디오에서 레이블을 감지하는 데 사용되는 레이블 감지 모델의 버전을 반환합니다.

포함 및 제외 필터

Amazon Rekognition Image 및 Amazon Rekognition Video 레이블 감지 작업에서 반환되는 결과를 필터링할 수 있습니다. 레이블 및 카테고리에 대한 필터링 기준을 제공하여 결과를 필터링하세요. 레이블 필터는 포함 또는 제외 필터일 수 있습니다.

`DetectLabels`를 사용하여 얻은 결과의 필터링에 대한 자세한 내용은 이미지에서 레이블 감지 섹션을 참조하세요.

GetLabelDetection으로 얻은 결과의 필터링에 대한 자세한 내용은 비디오에서 레이블 감지 섹션을 참조하세요.

결과 정렬 및 집계

특정 Amazon Rekognition Video 작업에서 얻은 결과를 타임스탬프 및 비디오 세그먼트에 따라 정렬하고 집계할 수 있습니다. 레이블 감지 또는 콘텐츠 조절 작업의 결과를 각각GetLabelDetection 또는 GetContentModeration으로 가져올 때는 SortBy 및 AggregateBy 인수를 사용하여 결과가 반환되는 방식을 지정할 수 있습니다. SortBy를 TIMESTAMP 또는 NAME(레이블 이름)과 함께 사용하고 TIMESTAMPS나 SEGMENTS를 AggregateBy 인수와 함께 사용할 수 있습니다.

이미지에서 레이블 감지

DetectLabels 작업을 사용하여 이미지에서 레이블(객체 및 개념)을 감지하고 이미지 속성에 대한 정보를 검색할 수 있습니다. 이미지 속성에는 전경색과 배경색, 이미지의 선명도, 밝기, 대비와 같은 속성이 포함됩니다. 이미지의 레이블만 검색하거나, 이미지의 속성만 검색하거나, 둘 다 검색할 수 있습니다. 예제는 Amazon S3 버킷에 저장된 이미지 분석 섹션을 참조하세요.

다음 예제에서는 AWS SDKs 및를 사용하여 AWS CLI 를 호출합니다DetectLabels. DetectLabels 작업 응답에 대한 자세한 내용은 DetectLabels 응답 단원을 참조하십시오.

이미지에서 레이블을 감지하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 나무, 집, 보트 등과 같은 객체가 한 개 이상 있는 이미지를 S3 버킷에 업로드합니다. 이미지는 .jpg 또는 .png 형식이어야 합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 예제를 사용하여 DetectLabels 작업을 호출합니다.

Java

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다.

```
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

이 예제는 `detect-labels` CLI 작업의 JSON 출력을 표시합니다. `bucket` 및 `photo`의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다. `profile-name`의 값을 개발자 프로필 이름으로 바꿉니다.

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

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
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

이 예제는 입력 이미지에서 감지된 레이블을 표시합니다. `main` 함수에서, `bucket` 및 `photo` 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

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

import boto3

def detect_labels(photo, bucket):

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

     response = client.detect_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}},
     MaxLabels=10,
     # Uncomment to use image properties and filtration settings
```

```
        #Features=["GENERAL_LABELS", "IMAGE_PROPERTIES"],
        #Settings={"GeneralLabels": {"LabelInclusionFilters":["Cat"]},
        # "ImageProperties": {"MaxDominantColors":10}}
        )

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

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

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

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

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

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

```
         print()

     return len(response['Labels'])

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

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

.NET

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다.

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

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

public class DetectLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
```

```
                    Bucket = bucket
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F
        };

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

Ruby

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다.

```
  # 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

이 예제는 입력 이미지에서 감지된 레이블 목록을 표시합니다. `bucket` 및 `photo`의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

TypeScript 정의를 사용하는 경우 Node.js로 프로그램을 실행하려면 `const AWS = require('aws-sdk')` 대신 `import AWS from 'aws-sdk'`를 사용해야 할 수도 있습니다. 자세한 내용은 AWS SDK for Javascript를 참조하세요. 구성 설정에 따라 `AWS.config.update({region:`*`region`*`});`으로 리전을 지정해야 할 수도 있습니다.

```
// 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//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 작업 요청

DetectLabel에 대한 입력은 이미지입니다. 이 예제 JSON 입력에서는 Amazon S3 버킷에서 소스 이미지를 불러옵니다. MaxLabels는 응답에 반환되는 레이블의 최대 수입니다. MinConfidence는 Amazon Rekognition Image가 감지된 레이블을 응답에 반환하기 위해 충족해야 하는 최소 정확성 신뢰도 수준입니다.

Features로 반환하려는 이미지의 특성을 하나 이상 지정할 수 있으며, 이를 통해 GENERAL_LABELS 및 IMAGE_PROPERTIES를 선택할 수 있습니다. GENERAL_LABELS를 포함하면 입력 이미지에서 감지된 레이블이 반환되고 IMAGE_PROPERTIES를 포함하면 이미지 색상 및 품질에 액세스할 수 있습니다.

Settings를 통해 반환된 항목을 GENERAL_LABELS 및 IMAGE_PROPERTIES 특성 둘 다에 대해 필터링할 수 있습니다. 레이블의 경우 포함 및 제외 필터를 사용할 수 있습니다. 특정 레이블, 개별 레이블 또는 레이블 카테고리별로 필터링할 수도 있습니다.

- LabelInclusionFilters - 응답에 포함할 레이블을 지정할 수 있습니다.
- LabelExclusionFilters - 응답에서 제외할 레이블을 지정할 수 있습니다.
- LabelCategoryInclusionFilters - 응답에 포함할 레이블 카테고리를 지정할 수 있습니다.
- LabelCategoryExclusionFilters - 응답에서 제외할 레이블 카테고리를 지정할 수 있습니다.

필요에 따라 포함 필터와 제외 필터를 조합하여 일부 레이블 또는 카테고리를 제외하는 동시에 다른 레이블이나 카테고리를 포함할 수도 있습니다.

IMAGE_PROPERTIES는 이미지의 주 색상 및 선명도, 밝기, 대비 등과 같은 품질 속성을 가리킵니다. IMAGE_PROPERTIES를 감지할 때 MaxDominantColors 파라미터를 사용하여 반환할 주 색상의 최대 수(기본값 10)를 지정할 수 있습니다.

```
{
    "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 응답

DetectLabels의 응답은 이미지에서 감지된 레이블의 배열과 각각의 해당 신뢰도 수준입니다.

다음은 DetectLabels의 응답 예제입니다. 아래 샘플 응답에는 다음을 포함하여 GENERAL_LABELS에 대해 반환되는 다양한 속성이 포함되어 있습니다.

- Name - 탐지된 레이블의 이름. 이 예제에서는 작업을 통해 Mobile Phone 레이블이 붙은 객체를 감지했습니다.
- Confidence - 각 레이블에는 연결된 신뢰도 수준이 있습니다. 이 예제에서 레이블에 대한 신뢰도는 99.36% 였습니다.
- Parents - 탐지된 레이블의 상위 레이블입니다. 이 예제에서 Mobile Phone 레이블에는 Phone이라는 상위 레이블이 있습니다.

- Aliases - 레이블에 있을 수 있는 별칭에 대한 정보. 이 예제에서 Mobile Phone 레이블에는 Cell Phone이라는 별칭이 있을 수 있습니다.
- Categories - 탐지된 레이블이 속하는 레이블 카테고리입니다. 이 예제에서는 Technology and Computing입니다.

일반적 객체 레이블에 대한 응답은 입력 이미지 상의 레이블 위치에 대한 경계 상자 정보를 포함합니다. 예를 들어 인물 레이블은 두 개의 경계 상자를 포함하는 인스턴스 배열을 갖습니다. 이들은 이미지에서 감지된 두 사람의 위치입니다.

응답에는 IMAGE_PROPERTIES와 관련된 속성도 포함됩니다. IMAGE_PROPERTIES 기능이 제공하는 속성은 다음과 같습니다.

- Quality - 입력 이미지의 선명도, 밝기 및 대비에 대한 정보로, 0에서 100 사이의 점수가 매겨집니다. 전체 이미지 및 가능한 경우 이미지의 배경 및 전경에 대한 품질이 보고됩니다. 그러나 대비는 전체 이미지에 대해서만 보고되는 반면 선명도 및 밝기는 배경 및 전경에 대해서도 보고됩니다.
- Dominant Color - 이미지의 주 색상 배열입니다. 각각의 주 색상은 단순화된 색상 이름, CSS 색상 팔레트, RGB 값 및 16진수 코드로 기술됩니다.
- Foreground - 입력 이미지 전경의 주 색상, 선명도 및 밝기에 대한 정보입니다.
- Background - 입력 이미지 배경의 주 색상, 선명도 및 밝기에 대한 정보입니다.

GENERAL_LABELS와 IMAGE_PROPERTIES를 입력 파라미터로 함께 사용하면 Amazon Rekognition Image는 경계 상자가 있는 객체의 주 색상도 반환합니다.

`LabelModelVersion` 필드에는 `DetectLabels`가 사용하는 감지 모델의 버전 번호가 포함됩니다.

```
{

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

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

```
                "CssColor": "fuscia",
                "PixelPercentage": 40.10
            }
        ],
        "Foreground": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "red",
                    "PixelPercentage": 30.70
                }
            ],
        }
        "Background": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "Red",
                    "PixelPercentage": 10.20
                }
            ],
        },
    },
    "LabelModelVersion": "3.0"
}
```

DetectLabels 응답의 변환

DetectLabels API를 사용하는 경우 기본 레이블과 별칭이 모두 동일한 목록에 포함된 이전 API 응답 구조를 모방하는 응답 구조가 필요할 수 있습니다.

다음은 DetectLabels의 현재 API 응답의 예입니다.

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

다음 예제는 DetectLabels API의 이전 응답을 보여줍니다.

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

```
            "Parents": [
                {
                "Name": "Phone"
                }
             ]
         },
]
```

필요한 경우 현재 응답을 이전 응답 형식을 따르도록 변환할 수 있습니다. 다음 샘플 코드를 사용하여 최신 API 응답을 이전 API 응답 구조로 변환할 수 있습니다.

Python

다음 코드 샘플은 DetectLabels API에서 현재 응답을 변환하는 방법을 보여줍니다. 아래 코드 샘플에서 *EXAMPLE_INFERENCE_OUTPUT* 값을 실행한 DetectLabels 작업의 출력으로 대체할 수 있습니다.

```
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)
```

다음은 변환된 응답의 예시입니다.

```
#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"
            ]
        }
    ]
}
```

비디오에서 레이블 감지

Amazon Rekognition Video는 비디오에서 레이블(객체 및 개념)과 레이블이 감지된 시간을 감지할 수 있습니다. SDK 코드에 대한 예는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석 (SDK) 단원을 참조하십시오. AWS CLI 예제는 단원을 참조하십시오를 사용하여 비디오 분석 AWS Command Line Interface.

Amazon Rekognition Video의 레이블 감지는 비동기 작업입니다. 비디오에서 레이블 감지를 시작하려면 StartLabelDetection을 직접 호출하세요.

Amazon Rekognition Video는 비디오 분석의 완료 상태를 Amazon Simple Notification Service 주제에 게시합니다. 비디오 분석이 성공적으로 완료되면 GetLabelDetection을 직접 호출하여 감지된 레이블을 가져옵니다. 비디오 분석 API 작업을 호출하는 방법에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오.

StartLabelDetection 요청

다음 예제는 `StartLabelDetection` 작업의 요청입니다. Amazon S3 버킷에 저장된 비디오를 사용하여 `StartLabelDetection` 작업을 제공합니다. 이 예제 요청 JSON에서는 Amazon S3 버킷 및 비디오 이름이, `MinConfidence`, `Features`, `Settings`, `NotificationChannel`와 함께 지정됩니다.

`MinConfidence`는 Amazon Rekognition Video가 감지된 레이블 또는 인스턴스 경계 상자(감지된 경우)를 응답에 반환하기 위해 가져야 할 최소 정확성 신뢰도입니다.

`Features`를 사용하면 GENERAL_LABELS가 응답의 일부로 반환되도록 지정할 수 있습니다.

`Settings`를 사용하면 GENERAL_LABELS에 대해 반환된 항목을 필터링할 수 있습니다. 레이블의 경우 포함 및 제외 필터를 사용할 수 있습니다. 특정 레이블, 개별 레이블 또는 레이블 카테고리별로 필터링할 수도 있습니다.

- `LabelInclusionFilters` - 응답에 포함할 레이블을 지정하는 데 사용됩니다.

- `LabelExclusionFilters` - 응답에서 제외할 레이블을 지정하는 데 사용됩니다.
- `LabelCategoryInclusionFilters` - 응답에 포함할 레이블 카테고리를 지정하는 데 사용됩니다.
- `LabelCategoryExclusionFilters` - 응답에서 제외할 레이블 카테고리를 지정하는 데 사용됩니다.

필요에 따라 포함 필터와 제외 필터를 조합하여 일부 레이블 또는 카테고리를 제외하는 동시에 다른 레이블이나 카테고리를 포함할 수도 있습니다.

`NotificationChannel`은 Amazon Rekognition Video에서 레이블 감지 작업의 완료 상태를 게시하려는 Amazon SNS 주제의 ARN입니다. `AmazonRekognitionServiceRole` 권한 정책을 사용하는 경우 Amazon SNS 주제의 이름은 반드시 Rekognition으로 시작해야 합니다.

다음은 필터를 포함한 JSON 형식의 샘플 `StartLabelDetection` 요청입니다.

```
{
    "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 작업 응답

GetLabelDetection은 비디오에서 감지된 레이블에 대한 정보가 포함된 배열(Labels)을 반환합니다. 배열을 시간별로 정렬하거나 SortBy 파라미터 지정할 때 감지된 레이블별로 정렬할 수 있습니다. AggregateBy 파라미터를 사용하여 응답 항목을 집계하는 방법을 선택할 수도 있습니다.

다음은 GetLabelDetection의 JSON 응답 예입니다. 응답에서 다음에 유의하십시오.

- Sort order - 시간별로 정렬된 반환된 레이블의 배열. 레이블별로 정렬하려면, GetLabelDetection에서 SortBy 입력 파라미터에 NAME를 지정하십시오. 레이블이 비디오에 여러 번 나오면 (LabelDetection)요소의 여러 인스턴스가 있습니다. 기본 정렬 순서는 TIMESTAMP이고 보조 정렬 순서는 NAME입니다.
- 레이블 정보 - LabelDetection 배열 요소에는 (Label)객체가 포함되어 있는데 이는 감지된의 레이블 이름과 Amazon Rekognition이 그에 대해 가지는 정확성 신뢰도를 포함합니다. 또한 Label 객체에는 레이블의 계층적 분류와 일반 레이블의 경우에는 경계 상자 정보가 포함됩니다. Timestamp는 비디오 시작부터 레이블이 감지될 때까지 지난 밀리초 단위의 시간입니다.

  레이블과 관련된 모든 카테고리 또는 별칭에 관한 정보도 반환됩니다. 비디오 SEGMENTS별로 집계된 결과의 경우 세그먼트의 시작 시간, 종료 시간, 지속 시간을 각각 정의하는 StartTimestampMillis, EndTimestampMillis, DurationMillis 구조가 반환됩니다.
- Aggregation - 결과가 반환될 때 집계되는 방식을 지정합니다. 기본값은 TIMESTAMPS 기준 집계입니다. 일정 기간 동안의 결과를 집계하는 SEGMENTS 집계 기준을 선택할 수도 있습니다. SEGMENTS 기준으로 집계할 경우 경계 상자가 있는 감지된 인스턴스에 대한 정보는 반환되지 않습니다. 세그먼트 내에서 감지된 레이블만 반환됩니다.
- 페이징 정보 - 이 예제는 레이블 감지 정보의 페이지 하나를 보여줍니다. GetLabelDetection의 MaxResults 입력 파라미터에서 반환하는 LabelDetection 객체의 개수를 지정할 수 있습니다. MaxResults 보다 많은 결과가 존재할 경우 GetLabelDetection은 결과의 다음 페이지를 가져올 때 사용되는 토큰(NextToken)을 반환합니다. 자세한 내용은 Amazon Rekognition Video 분석 결과 가져오기 단원을 참조하십시오.
- 비디오 정보 - 응답에는GetLabelDetection에서 반환된 정보의 각 페이지에 있는 비디오 형식(VideoMetadata)에 관한 정보가 포함되어 있습니다.

다음은 타임스탬프 기준으로 집계된 JSON 형식의 GetLabelDetection 응답 샘플입니다.

```
{
    "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
    }
}
```

다음은 세그먼트 기준으로 집계된 JSON 형식의 GetLabelDetection 응답 샘플입니다.

```
{
    "JobStatus": "SUCCEEDED",
    "LabelModelVersion": "3.0",
    "Labels": [
        {
            "StartTimestampMillis": 225,
            "EndTimestampMillis": 3578,
            "DurationMillis": 3353,
            "Label": {
                "Name": "Car",
                "Categories": [
                  {
                    "Name": "Vehicles and Automotive"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Automobile"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875 // Maximum confidence score for Segment
 mode
            }
        },
        {
            "StartTimestampMillis": 7578,
            "EndTimestampMillis": 12371,
            "DurationMillis": 4793,
```

```
            "Label": {
                "Name": "Kangaroo",
                "Categories": [
                  {
                    "Name": "Animals and Pets"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Wallaby"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Mammal"
                  }
                ],
                "Confidence": 99.9364013671875
            }
        },
        {
            "StartTimestampMillis": 22225,
            "EndTimestampMillis": 22578,
            "DurationMillis": 2353,
            "Label": {
                "Name": "Bicycle",
                "Categories": [
                  {
                    "Name": "Hobbies and Interests"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Bike"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875
            }
        }
```

```
    ],
    "VideoMetadata": {
        "ColorRange": "FULL",
        "DurationMillis": 5000,
        "Format": "MP4",
        "FrameWidth": 1280,
        "FrameHeight": 720,
        "FrameRate": 24
    }
}
```

GetLabelDetection 응답의 변환

GetLabelDetection API를 사용하여 결과를 가져오는 경우 기본 레이블과 별칭이 모두 동일한 목록에 포함된 이전 API 응답 구조를 모방하는 응답 구조가 필요할 수 있습니다.

이전 섹션에 있는 예제 JSON 응답에는 GetLabelDetection에서 오는 API 응답의 현재 형식이 표시됩니다.

다음 예제는 GetLabelDetection API의 이전 응답을 보여줍니다.

```
{
    "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
    }
}
```

필요한 경우 현재 응답을 이전 응답 형식을 따르도록 변환할 수 있습니다. 다음 샘플 코드를 사용하여 최신 API 응답을 이전 API 응답 구조로 변환할 수 있습니다.

```
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
```

스트리밍 비디오 이벤트의 레이블 감지

Amazon Rekognition Video를 사용하여 스트리밍 비디오에서 레이블을 감지할 수 있습니다. 이를 위해 스트림 프로세서(CreateStreamProcessor)를 생성하여 스트리밍 비디오 분석을 시작하고 관리합니다.

Amazon Rekognition Video는 Amazon Kinesis Video Streams를 사용하여 비디오 스트림을 수신하고 처리합니다. 스트림 프로세서를 생성할 때 스트림 프로세서가 감지할 항목을 선택합니다. 사람, 패키지, 반려동물 또는 사람과 패키지를 선택할 수 있습니다. 분석 결과는 Amazon S3 버킷과 Amazon SNS 알림에 출력됩니다. Amazon Rekognition Video는 동영상 속 사람의 존재를 감지하지만 그 사람이 특정 개인인지는 감지하지 못한다는 점에 유의하세요. 스트리밍 비디오의 컬렉션에서 얼굴을 검색하려면 the section called “스트리밍 비디오에서 컬렉션의 얼굴 검색” 섹션을 참조하세요.

스트리밍 비디오에 Amazon Rekognition Video를 사용하려면 애플리케이션에 다음 항목이 필요합니다.

- Amazon Rekognition Video로 스트리밍 비디오를 전송하기 위한 Kinesis 비디오 스트림. 자세한 내용은 Amazon Kinesis Video Streams 개발자 안내서를 참조하세요.

- 스트리밍 비디오 분석을 관리할 Amazon Rekognition Video 스트림 프로세서. 자세한 내용은 Amazon Rekognition Video 스트림 프로세서 작업 개요 단원을 참조하십시오.
- Amazon S3 버킷. Amazon Rekognition Video는 S3 버킷을 사용하여 세션 출력을 게시합니다. 출력 결과에는 관심 대상인 사람이나 객체가 처음으로 감지된 이미지 프레임이 포함됩니다. 귀하가 해당 S3 버킷의 소유자여야 합니다.
- Amazon Rekognition Video에서 스마트 알림과 세션 종료 요약을 게시하는 Amazon SNS 주제.

주제

- Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정
- 스트리밍 비디오 이벤트에 대한 레이블 감지 작업 직접 호출

Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정

다음 절차는 스트리밍 비디오에서 레이블을 감지하는 데 사용되는 Kinesis 비디오 스트림 및 기타 리소스를 프로비저닝하기 위해 수행할 단계를 설명합니다.

사전 조건

이 절차를 실행하려면 AWS SDK for Java 를 설치해야 합니다. 자세한 내용은 Amazon Rekognition 시작 단원을 참조하십시오. 사용할 AWS 계정은 Amazon Rekognition API에 대한 액세스 권한을 가지고 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 Amazon Rekognition에서 정의한 작업을 참조하세요.

비디오 스트림에서 레이블을 감지하려면(AWS SDK)

1. Amazon S3 버킷을 생성합니다. 사용하려는 버킷 이름과 키 접두사를 모두 기록해 둡니다. 나중에 이 정보가 필요합니다.
2. Amazon SNS 주제를 생성합니다. 이를 사용하여 관심 객체가 비디오 스트림에서 처음 감지될 때 알림을 수신할 수 있습니다. 해당 주제에 대한 Amazon 리소스 이름(ARN)을 기록해 두세요. 자세한 정보는 Amazon SNS 개발자 안내서의 Amazon SNS 주제 생성을 참조하세요.
3. 해당 Amazon SNS 주제의 엔드포인트를 구독합니다. 자세한 내용은 Amazon SNS 개발자 가이드의 Amazon SNS 주제 구독을 참조하세요.
4. Kinesis 비디오 스트림을 생성하고 스트림의 Amazon 리소스 이름(ARN)을 적어둡니다.
5. 아직 생성하지 않았다면, Amazon Rekognition Video에 Kinesis 비디오 스트림, S3 버킷 및 Amazon SNS 주제에 대한 액세스 권한을 부여하는 IAM 서비스 역할을 생성하세요. 자세한 내용은 레이블 감지 스트림 프로세서에 액세스 권한 부여 단원을 참조하십시오.

그 후 레이블 감지 스트림 프로세서를 생성하고 선택한 스트림 프로세서 이름을 사용하여 스트림 프로세서를 시작할 수 있습니다.

> **Note**
>
> 반드시 미디어를 Kinesis 비디오 스트림으로 수집할 수 있는지 확인한 후에 스트림 프로세서를 시작하세요.

카메라 방향 및 설정

Amazon Rekognition Video 스트리밍 비디오 이벤트는 Kinesis Video Streams에서 지원하는 모든 카메라를 지원할 수 있습니다. 최상의 결과를 얻으려면 카메라를 지면에서 0도에서 45도 사이에 두는 것이 좋습니다. 카메라는 표준 수직 위치에 있어야 합니다. 예를 들어 프레임 안에 사람이 있는 경우 사람은 수직으로 방향을 잡아야 하고 사람의 머리는 프레임에서 발보다 높아야 합니다.

레이블 감지 스트림 프로세서에 액세스 권한 부여

AWS Identity and Access Management (IAM) 서비스 역할을 사용하여 Amazon Rekognition Video에 Kinesis 비디오 스트림에 대한 읽기 액세스 권한을 부여합니다. 그러려면 IAM 역할을 사용하여 Amazon Rekognition Video에 Amazon S3 버킷 및 Amazon SNS 주제에 대한 액세스 권한을 부여하세요.

Amazon Rekognition Video에서 기존 Amazon SNS 주제, Amazon S3 버킷 및 Kinesis 비디오 스트림에 액세스할 수 있도록 허용하는 권한 정책을 생성할 수 있습니다. 를 사용하는 step-by-step 절차는 단원을 AWS CLI참조하십시오the section called “AWS CLI 레이블 감지 IAM 역할을 설정하는 명령”.

Amazon Rekognition Video에 레이블 감지를 위한 리소스에 대한 액세스 권한을 부여하려면

1. IAM JSON 정책 편집기로 새 권한 정책을 생성하고 다음 정책을 사용합니다. `topicarn`을 Kinesis 비디오 스트림의 이름으로, `kvs-stream-name`을 사용하려는 Amazon SNS 주제의 Amazon 리소스 이름(ARN)으로, `bucket-name`을 Amazon S3 버킷의 이름으로 바꾸세요.
2. IAM 서비스 역할을 생성하거나 기존 IAM 서비스 역할을 업데이트합니다. 다음 정보를 사용하여 IAM 서비스 역할을 생성하세요.

    1. 서비스 이름으로 Rekognition을 선택합니다.
    2. 서비스 역할 사용 사례로 [Rekognition]을 선택합니다.
    3. 1단계에 만든 권한 정책을 연결합니다.

3. 서비스 역할의 ARN을 기록합니다. 비디오 분석 작업을 수행하기 전에 스트림 프로세서를 생성해야 합니다.
4. (선택 사항) 자체 AWS KMS 키를 사용하여 S3 버킷으로 전송된 데이터를 암호화하는 경우 IAM 역할에 다음 문을 추가해야 합니다. (이는 사용하려는 고객 관리형 키에 대응하는 키 정책에 대해 생성한 IAM 역할입니다.)

```
            {
                        "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 레이블 감지 IAM 역할을 설정하는 명령

아직 하지 않았다면 자격 증명 AWS CLI 으로를 설정하고 구성합니다.

에 다음 명령을 입력하여 레이블 감지에 필요한 권한을 가진 IAM 역할을 AWS CLI 설정합니다.

1. `export IAM_ROLE_NAME=labels-test-role`
2. `export AWS_REGION=us-east-1`
3. 다음 내용이 포함된 신뢰 관계 정책 파일(예: assume-role-rekognition.json)을 생성합니다.

   JSON

```
{
  "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. 알림을 수신하고자 하는 SNS 주제의 이름이 “AmazonRekognition” 접두사로 시작하지 않는 경우 다음 정책을 추가하세요.

   `aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn "arn:aws:iam::aws:policy/AmazonSNSFullAccess" --region $AWS_REGION`
7. Amazon S3 버킷으로 전송되는 데이터를 암호화하는 데 자체 AWS KMS 키를 사용하는 경우 사용할 고객 관리형 키에 대한 키 정책을 업데이트하세요.

   a. 다음 콘텐츠가 포함된 kms_key_policy.json 파일을 생성하세요.

```
{
"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`. KMS_KEY_ID를 이전에 만든 KMS 키 ID로 바꾸세요.

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

스트리밍 비디오 이벤트에 대한 레이블 감지 작업 직접 호출

Amazon Rekognition Video는 스트리밍 비디오에서 사람이나 관련 객체를 감지하여 사용자에게 알릴 수 있습니다. 레이블 감지 스트림 프로세서를 생성할 때 Amazon Rekognition Video에서 탐지할 레이블을 선택하세요. 이는 사람, 패키지, 반려동물 또는 사람, 패키지 및 반려동물이 될 수 있습니다. 감지하고자 하는 레이블만 선택하세요. 이렇게 하면 해당하는 레이블만 알림을 생성합니다. 비디오 정보를 저장할 시기를 결정하는 옵션을 구성한 다음 프레임에서 감지된 레이블을 기반으로 추가 처리를 수행할 수 있습니다.

리소스를 설정한 후 스트리밍 비디오에서 레이블을 감지하는 프로세스는 다음과 같습니다.

1. 스트림 프로세서 생성
2. 스트림 프로세서 시작
3. 관심 객체가 감지되면 각 관심 객체를 처음 감지했을 때 Amazon SNS 알림을 받게 됩니다.
4. `MaxDurationInSeconds`에 지정된 시간이 지나면 스트림 프로세서가 중지됩니다.
5. 이벤트 요약이 포함된 최종 Amazon SNS 알림을 받게 됩니다.
6. Amazon Rekognition Video가 S3 버킷에 자세한 세션 요약을 게시합니다.

주제

- Amazon Rekognition Video 레이블 감지 스트림 프로세서 생성
- Amazon Rekognition Video 레이블 감지 스트림 프로세서 시작
- 레이블 감지 결과 분석

Amazon Rekognition Video 레이블 감지 스트림 프로세서 생성

스트리밍 비디오를 분석하기 전에 Amazon Rekognition Video 스트림 프로세서를 만듭니다 (CreateStreamProcessor).

관심 레이블과 사람을 감지하는 스트림 프로세서를 생성하려면 Kinesis 비디오 스트림(`Input`), Amazon S3 버킷 정보(`Output`) 및 Amazon SNS 주제

ARN(StreamProcessorNotificationChannel)을 입력으로 제공하세요. KMS 키 ID를 제공하여 S3 버킷으로 전송된 데이터를 암호화할 수도 있습니다. 사람, 패키지 및 사람, 또는 반려동물, 사람, 패키지 등 Settings에서 감지할 항목을 지정합니다. Amazon Rekognition에서 RegionsOfInterest로 모니터링할 프레임 내의 위치를 지정할 수도 있습니다. 다음은 CreateStreamProcessor 요청에 대한 JSON 예제입니다.

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

```
        "Width": 0.33,
        "Height": 0.44
      }
    },
    {
      "Polygon": [
        {
          "X": 0.11,
          "Y": 0.11
        },
        {
          "X": 0.22,
          "Y": 0.22
        },
        {
          "X": 0.33,
          "Y": 0.33
        }
      ]
    }
  ]
}
```

참고로 스트림 프로세서에 ConnectedHomeSettings를 지정하면 MinConfidence 값을 변경할 수 있습니다. MinConfidence는 알고리즘의 예측이 얼마나 확실한지를 나타내는 0에서 100 사이의 숫자 값입니다. 예를 들어, 신뢰값이 90인 person에 대한 알림은 알고리즘이 동영상에 사람이 있다는 것을 절대적으로 확신한다는 것을 의미합니다. 신뢰도 값이 10이면 사람이 있을 수도 있음을 나타냅니다. 알림을 수신하고 싶은 빈도에 따라 MinConfidence를 0에서 100 사이에서 원하는 값으로 설정할 수 있습니다. 예를 들어 Rekognition에서 비디오 프레임에 패키지가 있다고 확신할 때만 알림을 받으려면 MinConfidence를 90으로 설정할 수 있습니다.

기본적으로 MinConfidence 는 50으로 설정됩니다. 더 높은 정밀도를 위해 알고리즘을 최적화하려는 경우 MinConfidence를 50보다 높게 설정할 수 있습니다. 그러면 수신되는 알림은 줄어들지만 각 알림의 신뢰성은 더 높아집니다. 더 높은 재현율을 위해 알고리즘을 최적화하려는 경우 더 많은 알림을 수신하도록 MinConfidence를 50보다 낮게 설정할 수 있습니다.

Amazon Rekognition Video 레이블 감지 스트림 프로세서 시작

CreateStreamProcessor에서 지정한 스트림 프로세서 이름으로 StartStreamProcessor를 직접 호출하여 스트리밍 비디오 분석을 시작합니다. 레이블 감지 스트림 프로세서에서

StartStreamProcessor 작업을 실행할 때는 시작 및 중지 정보를 입력하여 처리 시간을 결정합니다.

스트림 프로세서를 시작하면 레이블 감지 스트림 프로세서 상태가 다음과 같이 변경됩니다.

1. StartStreamProcessor를 직접 호출하면 레이블 감지 스트림 프로세서 상태가 STOPPED 또는 FAILED에서 STARTING으로 바뀝니다.
2. 레이블 감지 스트림 프로세서가 실행되는 동안에는 상태가 STARTING으로 유지됩니다.
3. 레이블 감지 스트림 프로세서 실행이 완료되면 상태는 STOPPED 또는 FAILED가 됩니다.

StartSelector는 Kinesis 스트림에서 처리를 시작할 시작점을 지정합니다. KVS 생산자 타임스탬프 또는 KVS 조각 번호를 사용할 수 있습니다. 자세한 내용은 조각을 참조하세요.

> **Note**
>
> KVS 생산자 타임스탬프를 사용하는 경우 시간을 밀리초 단위로 입력해야 합니다.

StopSelector는 스트림 처리를 중지하는 시점을 지정합니다. 비디오를 처리할 최대 시간을 지정할 수 있습니다. 기본값은 최대 시간 10초입니다. 개별 KVS 조각의 크기에 따라 실제 처리 시간이 최대 처리 시간보다 약간 더 길 수 있다는 점에 유의하세요. 조각의 끝에서 최대 시간에 도달했거나 초과하면 처리 시간이 중지됩니다.

다음은 StartStreamProcessor 요청에 대한 JSON 예제입니다.

```
{
   "Name": "string",
   "StartSelector": {
     "KVSStreamStartSelector": {
         "KVSProducerTimestamp": 1655930623123
      },
        "StopSelector": {
            "MaxDurationInSeconds": 11
      }
   }
}
```

스트림 프로세서가 성공적으로 시작되면 HTTP 200 응답이 반환됩니다. 빈 JSON 본문이 포함됩니다.

레이블 감지 결과 분석

Amazon Rekognition Video가 레이블 탐지 스트림 프로세서로부터 알림을 게시하는 방법에는 세 가지가 있습니다. 객체 감지 이벤트에 대한 Amazon SNS 알림, 세션 종료 요약에 대한 Amazon SNS 알림, 상세한 Amazon S3 버킷 보고서입니다.

- 객체 감지 이벤트에 대한 Amazon SNS 알림

  비디오 스트림에서 레이블이 감지되면 객체 탐지 이벤트에 대한 Amazon SNS 알림을 받게 됩니다. Amazon Rekognition은 관심 객체나 사람이 비디오 스트림에서 처음 감지될 때 알림을 게시합니다. 알림에는 감지된 레이블 유형, 신뢰도, 히어로 이미지로 연결되는 링크 등의 정보가 포함됩니다. 또한 감지된 사람이나 객체의 잘라낸 이미지와 감지 타임스탬프도 포함됩니다. 알림은 다음 형식을 취합니다.

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

```
                "width": number
            }
        }],
        "eventId": string,
        "tags": {
            [string]: string
        },
        "sessionId": string,
        "startStreamProcessorRequest": object
    }
}
```

- Amazon SNS 세션 종료 요약

  스트림 처리 세션이 완료되면 Amazon SNS 알림도 수신하게 됩니다. 이 알림에는 세션에 대한 메타데이터가 나열됩니다. 여기에는 처리된 스트림의 기간과 같은 세부 정보가 포함됩니다. 알림은 다음 형식을 취합니다.

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

- Amazon S3 버킷 보고서

  Amazon Rekognition Video는 `CreateStreamProcessor` 작업에서 제공된 Amazon S3 버킷에 비디오 분석 작업의 자세한 추론 결과를 게시합니다. 이 결과에는 관심 대상인 객체나 사람이 처음으로 감지된 이미지 프레임이 포함됩니다.

  해당 프레임은 S3의 ObjectKeyPrefix/StreamProcessorName/SessionId/*`service_determined_unique_path`* 경로에서 찾을 수 있습니다. 이 경로에서 LabelKeyPrefix는 고객이 제공한 선택적 인수이고, StreamProcessorName은 스트림 프로세서 리소스의 이름이며, SessionId는 스트림 처리 세션의 고유 ID입니다. 상황에 따라 대체하면 됩니다.

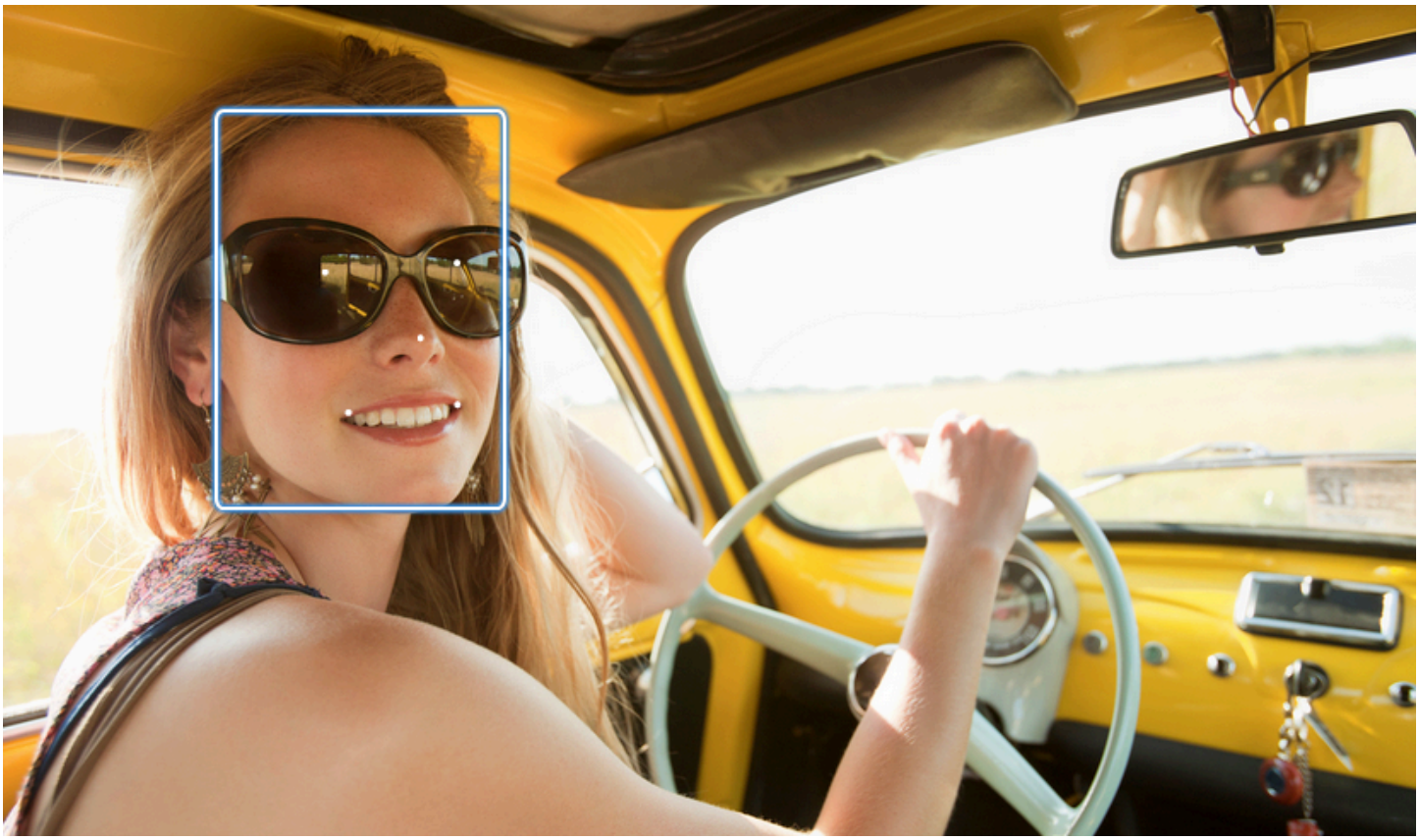
사용자 지정 레이블 감지

Amazon Rekognition Custom Labels는 이미지에서 비즈니스 요구 사항과 관련된 객체와 장면(예: 로고 또는 엔지니어링 기계 부품)을 식별할 수 있습니다. 자세한 내용은 Amazon Rekognition Custom Labels 개발자 안내서의 Amazon Rekognition Custom Labels란 무엇인가요?를 참조하세요.

얼굴 감지 및 분석

Amazon Rekognition은 이미지 및 비디오의 얼굴을 감지하고 분석하는 데 사용할 수 있는 API를 제공합니다. 이 섹션에서는 얼굴 분석을 위한 비스토리지 작업에 대한 개요를 제공합니다. 이러한 작업에는 얼굴 표식 감지, 감정 분석, 얼굴 비교와 같은 기능이 포함됩니다.

Amazon Rekognition은 얼굴 표식(예: 눈 위치)을 식별하고, 감정(예: 행복 또는 슬픔) 및 기타 속성(예: 안경 유무, 얼굴 가려짐)을 감지할 수 있습니다. 얼굴이 감지되면 시스템은 얼굴 속성을 분석하고 각 속성에 대한 신뢰도 점수를 반환합니다.



이 섹션에는 이미지 및 비디오 작업에 대한 예제가 포함되어 있습니다.

Rekognition의 이미지 작업 사용에 대한 자세한 내용은 이미지 작업 섹션을 참조하세요.

Rekognition의 비디오 작업 사용에 대한 자세한 내용은 저장된 비디오 분석 작업 섹션을 참조하세요.

이러한 작업은 비스토리지 작업입니다. 스토리지 작업 및 얼굴 컬렉션을 사용하여 이미지에서 감지된 얼굴 메타데이터를 저장할 수 있습니다. 나중에 이미지와 비디오에 저장된 얼굴을 검색할 수 있습니다. 예를 들어, 이것을 이용해 비디오에서 특정 인물을 검색할 수 있습니다. 자세한 내용은 컬렉션에서 얼굴 검색 단원을 참조하십시오.

자세한 내용은 Amazon Rekognition FAQ의 얼굴 섹션을 참조하세요.

> **Note**
>
> Amazon Rekognition Image 및 Amazon Rekognition Video에서 사용되는 얼굴 감지 모델은 만화나 애니메이션 캐릭터나 사람이 아닌 개체의 얼굴 감지를 지원하지 않습니다. 이미지 또는 비디오에서 만화 캐릭터를 감지하려면 Amazon Rekognition Custom Labels를 사용하는 것이 좋습니다. 자세한 내용은 Amazon Rekognition Custom Labels 개발자 안내서를 참조하세요.

주제

- 얼굴 감지 및 얼굴 비교 개요
- 얼굴 속성에 대한 가이드라인
- 이미지에서 얼굴 감지
- 이미지에 있는 얼굴 비교
- 저장된 비디오에서 얼굴 감지

얼굴 감지 및 얼굴 비교 개요

Amazon Rekognition을 사용하면 얼굴이 포함된 이미지에 대한 두 가지 기본 기계 학습 애플리케이션인 얼굴 감지 및 얼굴 비교에 액세스할 수 있습니다. 얼굴 분석 및 ID 확인과 같은 중요한 기능을 강화하여 보안부터 개인 사진 조직에 이르기까지 다양한 애플리케이션에 필수적입니다.

얼굴 인식

얼굴 감지 시스템은 '이 사진에 얼굴이 있는가?'라는 질문을 해결합니다. 얼굴 감지의 주요 측면은 다음과 같습니다.

- 위치 및 방향: 이미지 또는 비디오 프레임에서 얼굴의 존재 유무, 위치, 크기 및 방향을 결정합니다.
- 얼굴 속성: 성별, 연령 또는 수염과 같은 속성에 관계없이 얼굴을 감지합니다.
- 추가 정보: 얼굴 가려짐 및 시선 방향에 대한 세부 정보를 제공합니다.

얼굴 비교

얼굴 비교 시스템은 '한 이미지 안의 얼굴이 다른 이미지의 얼굴과 일치하는가?'라는 질문에 초점을 둡니다. 얼굴 비교 시스템 기능은 다음과 같습니다.

- 얼굴 일치 예측: 이미지의 얼굴을 제공된 데이터베이스의 얼굴과 비교하여 일치를 예측합니다.
- 얼굴 속성 처리: 표정, 수염, 연령에 관계없이 얼굴을 비교할 수 있도록 속성을 처리합니다.

신뢰도 점수 및 감지 누락

얼굴 감지 및 얼굴 비교 시스템 모두 신뢰도 점수를 활용합니다. 신뢰도 점수는 얼굴의 존재 또는 얼굴 간 일치와 같은 예측 가능성을 나타냅니다. 점수가 높을수록 가능성이 높음을 나타냅니다. 예를 들어 90% 신뢰도는 60%보다 정확한 감지 또는 일치 확률이 높음을 의미합니다.

얼굴 감지 시스템이 적절하게 얼굴을 감지하지 못하거나 실제 얼굴에 대해 낮은 신뢰도의 예측을 제공할 경우 이를 감지 누락/미검이라고 합니다. 시스템이 높은 신뢰도에서 얼굴의 존재를 잘못 예측할 경우 이를 오경보/오탐이라고 합니다.

마찬가지로, 얼굴 비교 시스템이 같은 사람의 두 얼굴을 일치시키지 못할 수 있으며(감지 누락/미검) 다른 사람의 두 얼굴을 같은 사람이라고 잘못 예측할 수 있습니다(오경보/오탐).

애플리케이션 설계 및 임곗값 설정

- 결과를 반환하는 데 필요한 최소 신뢰도 수준을 지정하는 임곗값을 설정할 수 있습니다. 적절한 신뢰도 임곗값을 선택하는 것은 시스템 출력에 기반한 애플리케이션 설계 및 의사 결정에 필수적입니다.
- 선택한 신뢰도 수준은 사용 사례를 반영해야 합니다. 사용 사례 및 신뢰도 임곗값의 몇 가지 예:
- • 사진 애플리케이션: 사진에서 가족을 식별하는 데는 낮은 임곗값(예: 80%)으로도 충분할 수 있습니다.
  - 고위험 시나리오: 보안 애플리케이션과 같이 감지 누락 또는 오경보의 위험이 더 높은 사용 사례에서는 시스템이 더 높은 신뢰도를 사용해야 합니다. 이러한 경우 정확한 얼굴 일치를 위해 더 높은 임곗값(예: 99%)이 권장됩니다.

신뢰도 임곗값 설정 및 이해에 대한 자세한 내용은 컬렉션에서 얼굴 검색 섹션을 참조하세요.

얼굴 속성에 대한 가이드라인

다음은 Amazon Rekognition이 얼굴 속성을 처리하고 반환하는 방법에 대한 세부 정보입니다.

- FaceDetail 객체: 감지된 각 얼굴에 대해 FaceDetail 객체가 반환됩니다. 이 FaceDetail에는 얼굴 표식, 품질, 포즈 등에 대한 데이터가 포함되어 있습니다.
- 속성 예측: 감정, 성별, 연령 등의 속성이 예측됩니다. 각 예측에 신뢰도 수준이 할당되고, 예측은 해당 신뢰도 점수와 함께 반환됩니다. 민감한 사용 사례에는 99% 신뢰도 임곗값이 권장됩니다. 연령 추정의 경우 예측된 연령 범위의 중간점이 최상의 근사치를 제공합니다.

성별 및 감정 예측은 외양적 모습을 기반으로 하며 실제 성 정체성 또는 감정 상태를 결정하는 데 사용해서는 안 됩니다. 성별 이분법(남성/여성) 예측은 특정 이미지에서 얼굴의 외양적 모습을 기반으로 합니다. 이는 사람의 성 정체성을 나타내지 않으므로 그러한 결정을 내리는 데 Rekognition을 사용하지 않아야 합니다. 개인의 권리, 사생활 또는 서비스 액세스에 영향을 미칠 수 있는 결정을 내리는 데에는 성별 이분법적 예측을 사용하지 않는 것을 권장합니다. 마찬가지로, 감정의 예측은 사람 내면의 실제 감정 상태를 나타내지 않으므로 그러한 결정을 내리는 데 Rekognition을 사용하지 않아야 합니다. 사진에서 행복한 얼굴을 하고 있는 것처럼 꾸민 사람이 행복해 보일 수는 있지만 행복한 감정을 느끼고 있지 않을 수도 있습니다.

애플리케이션 및 사용 사례

다음은 이러한 속성에 대한 몇 가지 실용적인 애플리케이션 및 사용 사례입니다.

- 애플리케이션: 미소, 포즈, 선명도와 같은 속성을 사용하여 프로필 사진을 선택하거나 익명으로 인구 통계를 추정할 수 있습니다.
- 일반적인 사용 사례: 이벤트 또는 소매점의 소셜 미디어 애플리케이션 및 인구 통계 추정이 일반적인 예입니다.

각 속성에 대한 자세한 내용은 FaceDetail을 참조하세요.

이미지에서 얼굴 감지

Amazon Rekognition Image는 입력 이미지에서 눈, 코, 입 등 주요 얼굴 특성을 찾는 DetectFaces 작업을 제공합니다. Amazon Rekognition Image는 이미지에서 가장 큰 얼굴 100개를 감지합니다.

입력 이미지를 이미지 바이트 배열(base64 인코딩 이미지 바이트)로 제공하거나 Amazon S3 객체를 지정할 수 있습니다. 이 절차에서는 S3 버킷에 이미지(JPEG 또는 PNG)를 업로드하고 객체 키 이름을 지정합니다.

이미지에서 얼굴 감지

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. `AmazonRekognitionFullAccess` 권한과 `AmazonS3ReadOnlyAccess` 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 한 개 이상의 얼굴이 포함된 이미지를 S3 버킷에 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 예제를 사용하여 DetectFaces를 호출합니다.

   Java

   이 예제에서는 감지된 얼굴의 추정 연령 범위를 표시하고 감지된 모든 얼굴 속성의 JSON을 나열합니다. photo의 값을 이미지 파일 이름으로 변경합니다. amzn-s3-demo-bucket의 값을 이미지가 저장된 Amazon S3 버킷으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
import java.util.List;

//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.AgeRange;

//snippet-end:[rekognition.java2.detect_labels.import]

public class DetectFaces {

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage: " +
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,amzn-s3-demo-bucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

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

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
```

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

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

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

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

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

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

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

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

        } catch (RekognitionException e) {
```

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

AWS CLI

이 예제에서는 detect-faces AWS CLI 작업의 JSON 출력을 표시합니다. file을 이미지 파일의 이름으로 바꿉니다. amzn-s3-demo-bucket을 이미지 파일이 들어 있는 Amazon S3 버킷의 이름으로 바꿉니다.

```
aws rekognition detect-faces --image "{"S3Object":{"Bucket":"amzn-s3-demo-
bucket,"Name":"image-name"}}"\
                             --attributes "ALL" --profile profile-name --region
 region-name
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
aws rekognition detect-faces --image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --attributes "ALL"
--profile profile-name --region region-name
```

Python

이 예제에서는 감지된 얼굴의 추정 연령 범위 및 기타 속성을 표시하고 감지된 모든 얼굴 속성의 JSON을 나열합니다. photo의 값을 이미지 파일 이름으로 변경합니다. amzn-s3-demo-bucket의 값을 이미지가 저장된 Amazon S3 버킷으로 변경합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
import boto3
import json

def detect_faces(photo, bucket, region):
```

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

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

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

        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))

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

    return len(response['FaceDetails'])

def main():
    photo='photo'
    bucket='amzn-s3-demo-bucket'
    region='region'
    face_count=detect_faces(photo, bucket, region)
    print("Faces detected: " + str(face_count))

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

.NET

이 예제에서는 감지된 얼굴의 추정 연령 범위를 표시하고 감지된 모든 얼굴 속성의 JSON을 나열합니다. `photo`의 값을 이미지 파일 이름으로 변경합니다. `amzn-s3-demo-bucket`의 값을 이미지가 저장된 Amazon S3 버킷으로 변경합니다.

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

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

public class DetectFaces
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

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

        try
        {
            DetectFacesResponse detectFacesResponse =
 rekognitionClient.DetectFaces(detectFacesRequest);
            bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
            foreach(FaceDetail face in detectFacesResponse.FaceDetails)
            {
```

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

Ruby

이 예제에서는 감지된 얼굴의 추정 연령 범위를 표시하고, 감지된 모든 얼굴 속성을 나열합니다. photo의 값을 이미지 파일 이름으로 변경합니다. amzn-s3-demo-bucket의 값을 이미지가 저장된 Amazon S3 버킷으로 변경합니다.

```
  # 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

이 예제에서는 감지된 얼굴의 추정 연령 범위를 표시하고, 감지된 모든 얼굴 속성을 나열합니다. photo의 값을 이미지 파일 이름으로 변경합니다. amzn-s3-demo-bucket의 값을 이미지가 저장된 Amazon S3 버킷으로 변경합니다.

Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

TypeScript 정의를 사용하는 경우 Node.js로 프로그램을 실행하려면 `const AWS = require('aws-sdk')` 대신 `import AWS from 'aws-sdk'`를 사용해야 할 수도 있습니다. 자세한 내용은 AWS SDK for Javascript를 참조하세요. 구성 설정에 따라 `AWS.config.update({region:region});`으로 리전을 지정해야 할 수도 있습니다.

```
// 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 작업 요청

DetectFaces에 대한 입력은 이미지입니다. 이 예제에서는 Amazon S3 버킷에서 이미지를 불러옵니다. Attributes 파라미터는 모든 얼굴 속성을 반환하도록 지정합니다. 자세한 내용은 이미지 작업 단원을 참조하십시오.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "input.jpg"
        }
    },
    "Attributes": [
        "ALL"
    ]
}
```

DetectFaces 작업 응답

DetectFaces는 감지된 각 얼굴에 대해 다음 정보를 반환합니다.

- 경계 상자 - 얼굴 주위를 두르는 경계 상자의 좌표.
- 신뢰도 - 경계 상자에 얼굴이 포함될 신뢰도 수준.
- 얼굴 표식 – 얼굴 표식의 배열. 응답은 왼쪽 눈, 오른쪽 눈, 입 같은 각각의 표식의 x, y 좌표를 제공합니다.
- 얼굴 속성 - FaceDetail 객체로 반환되는, 얼굴의 가려짐 여부와 같은 얼굴 속성의 집합. 이 집합에는 AgeRange, Beard, Emotions, EyeDirection, Eyeglasses, EyesOpen, FaceOccluded, Gender, MouthOpen, Mustache, Smile, Sunglasses가 포함됩니다. 응답은 각 속성의 값을 제공합니다. 이 값은 부울(사람이 선글라스를 착용하고 있는지 아닌지), 문자열(남성인지 여성인지), 각도값(시선 방향의 피치나 요) 등 다양한 유형이 될 수 있습니다. 또한 대부분의 속성의 경우, 응답은 해당 속성에 대해 감지된 값의 신뢰도도 제공합니다. 단, FaceOccluded 속성과 EyeDirection 속성은

DetectFaces를 사용할 때는 지원되지만, StartFaceDetection 및 GetFaceDetection을 사용하여 비디오를 분석할 때는 지원되지 않습니다.

- 품질 – 얼굴의 밝기와 선명도를 기술합니다. 최상의 얼굴 감지을 보장하는 것에 관한 내용은 얼굴 비교 입력 이미지에 대한 권장 사항 단원을 참조하십시오.
- 포즈 – 이미지 내 얼굴의 회전을 기술합니다.

요청에는 반환하려는 얼굴 속성의 배열이 표시될 수 있습니다. 얼굴 속성의 DEFAULT 하위 집합(BoundingBox, Confidence, Pose, Quality, Landmarks)은 항상 반환됩니다. ["DEFAULT", "FACE_OCCLUDED", "EYE_DIRECTION"]를 사용하여 특정 얼굴 속성(기본 목록에 더해) 여러 개나 ["FACE_OCCLUDED"]와 같은 단일 속성의 반환을 요청할 수 있습니다. ["ALL"]을 사용하여 모든 얼굴 속성의 반환을 요청할 수 있습니다. 더 많은 속성을 요청하면 응답 시간이 늘어날 수 있습니다.

다음은 DetectFaces API 직접 호출 응답의 예입니다.


```
{
  "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
  }
}
```


다음 사항에 유의하세요.

- Pose 데이터는 감지된 얼굴의 회전을 기술합니다. BoundingBox와 Pose 데이터의 조합을 사용하여 애플리케이션이 표시하는 얼굴 주위에 경계 상자를 그릴 수 있습니다.
- Quality는 얼굴의 밝기와 선명도를 기술합니다. 이는 여러 이미지에서 얼굴을 비교해 가장 좋은 얼굴을 찾는 데 유용할 수 있습니다.
- 이전 응답은 서비스가 감지할 수 있는 모든 얼굴 landmarks, 모든 얼굴 속성 및 감정을 보여줍니다. 응답에서 이 모두를 얻으려면 attributes 파라미터를 값 ALL로 지정해야 합니다. DetectFaces API는 기본적으로 BoundingBox, Confidence, Pose, Quality, landmarks와 같은 5가지 얼굴 속성만 반환합니다. 기본 표식은 eyeLeft, eyeRight, nose, mouthLeft, mouthRight가 반환됩니다.

이미지에 있는 얼굴 비교

Rekognition을 사용하면 CompareFaces 작업을 사용하여 두 이미지 간의 얼굴을 비교할 수 있습니다. 이 기능은 ID 확인 또는 사진 매칭과 같은 애플리케이션에 유용합니다.

CompareFaces는 소스 이미지에 있는 얼굴을 대상 이미지에 있는 각 얼굴과 비교합니다. 이미지는 다음 중 하나로 CompareFaces에 전달됩니다.

- 이미지의 base64 인코딩 표시.

- Amazon S3 객체.

얼굴 감지와 얼굴 비교

얼굴 비교는 얼굴 감지와는 다릅니다. 얼굴 감지(DetectFaces 사용)는 이미지 또는 비디오에서 얼굴의 존재 여부 및 위치만 식별합니다. 반면 얼굴 비교에는 소스 이미지에서 감지된 얼굴을 대상 이미지의 얼굴과 비교하여 일치하는 항목을 찾는 작업이 포함됩니다.

유사성 임곗값

`similarityThreshold` 파라미터를 사용하여 응답에 포함할 일치 항목의 최소 신뢰도 수준을 정의합니다. 기본적으로 유사성 점수가 80% 이상인 얼굴만 응답에 반환됩니다.

> **Note**
>
> `CompareFaces`는 확률론적인 기계 학습 알고리즘을 사용합니다. False negative는 대상 이미지의 얼굴이 소스 이미지의 얼굴과 비교할 때 유사성 신뢰도 점수가 낮다는 잘못된 예측입니다. False negative의 확률을 줄이려면 대상 이미지를 여러 소스 이미지와 비교하는 것이 좋습니다. `CompareFaces`를 사용하여 개인의 권리, 개인 정보 보호 또는 서비스 액세스에 영향을 미치는 결정을 내리려는 경우 조치를 취하기 전에 결과를 사람에게 전달하여 검토 및 추가 검증을 받는 것을 권고합니다.

다음 코드 예제에서는 다양한 AWS SDKs에 CompareFaces 작업을 사용하는 방법을 보여줍니다. 이 AWS CLI 예제에서는 Amazon S3 버킷에 두 개의 JPEG 이미지를 업로드하고 객체 키 이름을 지정합니다. 다른 예제에서는 로컬 파일 시스템에서 파일 2개를 로드하여 이미지 바이트 배열로 입력합니다.

얼굴을 비교하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. `AmazonRekognitionFullAccess` 및 `AmazonS3ReadOnlyAccess` (AWS CLI 예제만 해당) 권한을 사용하여 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제 코드를 사용하여 `CompareFaces` 작업을 호출하십시오.

Java

이 예제는 로컬 파일 시스템에서 불러오는 소스 이미지와 대상 이미지의 일치 얼굴에 대한 정보를 표시합니다.

sourceImage 값과 targetImage 값을 소스 및 대상 이미지의 파일 이름과 경로로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
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

이 예제에서는 compare-faces AWS CLI 작업의 JSON 출력을 표시합니다.

amzn-s3-demo-bucket을 소스 이미지와 대상 이미지가 들어 있는 Amazon S3 버킷의 이름으로 바꿉니다. source.jpg와 target.jpg를 소스 이미지와 대상 이미지의 파일 이름으로 바꿉니다.

```
aws rekognition compare-faces --target-image \
"{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-name"}}" \
--source-image "{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-
name"}}"
--profile profile-name
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
aws rekognition compare-faces --target-image "{\"S3Object\":{\"Bucket\":\"amzn-
s3-demo-bucket\",\"Name\":\"image-name\"}}" \
--source-image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":
\"image-name\"}}" --profile profile-name
```

Python

이 예제는 로컬 파일 시스템에서 불러오는 소스 이미지와 대상 이미지의 일치 얼굴에 대한 정보를 표시합니다.

source_file 값과 target_file 값을 소스 및 대상 이미지의 파일 이름과 경로로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 로컬 파일 시스템에서 불러오는 소스 이미지와 대상 이미지의 일치 얼굴에 대한 정보를 표시합니다.

`sourceImage` 값과 `targetImage` 값을 소스 및 대상 이미지의 파일 이름과 경로로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 로컬 파일 시스템에서 불러오는 소스 이미지와 대상 이미지의 일치 얼굴에 대한 정보를 표시합니다.

photo_source 값과 photo_target 값을 소스 및 대상 이미지의 파일 이름과 경로로 바꿉니다.

```
 # 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

이 예제는 로컬 파일 시스템에서 불러오는 소스 이미지와 대상 이미지의 일치 얼굴에 대한 정보를 표시합니다.

photo_source 값과 photo_target 값을 소스 및 대상 이미지의 파일 이름과 경로로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
// 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 작업 요청

CompareFaces에 대한 입력은 이미지입니다. 이 예제에서는 소스 이미지와 대상 이미지를 로컬 파일 시스템에서 불러옵니다. SimilarityThreshold 입력 파라미터는 비교 얼굴을 응답에 포함시키는 기준이 되는 최소 신뢰도를 지정합니다. 자세한 내용은 이미지 작업 단원을 참조하십시오.

```
{
    "SourceImage": {
        "Bytes": "/9j/4AAQSk2Q==..."
    },
    "TargetImage": {
        "Bytes": "/9j/4O1Q==..."
    },
    "SimilarityThreshold": 70
}
```

CompareFaces 작업 응답

응답에는 다음이 포함됩니다.

- 얼굴 일치 배열: 일치하는 각 얼굴에 대한 유사성 점수와 메타데이터가 포함된 일치하는 얼굴 목록입니다. 여러 얼굴이 일치하는 경우 faceMatches

  배열에는 모든 얼굴 일치 항목이 포함됩니다.
- 얼굴 일치 세부 정보: 일치하는 각 얼굴은 경계 상자, 신뢰도 값, 표식 위치 및 유사성 점수도 제공됩니다.
- 일치하지 않는 얼굴 목록: 응답에는 소스 이미지 얼굴과 일치하지 않는 대상 이미지의 얼굴도 포함됩니다. 비교할 수 없는 각 얼굴에 대한 경계 상자를 포함합니다.
- 원본 얼굴 정보: 경계 상자와 신뢰도 값을 포함해 비교에 사용했던 원본 이미지의 얼굴에 대한 정보가 포함됩니다.

예제는 대상 이미지에서 찾은 하나의 얼굴 일치를 보여줍니다. 이 얼굴 일치의 경우, 경계 상자와 신뢰도 값(Amazon Rekognition이 경계 상자에 얼굴이 포함되었다고 믿는 신뢰도 수준)이 제공됩니다. 99.99의 유사성 점수는 얼굴들이 얼마나 유사한지를 나타냅니다. 또한 이 예제는 Amazon Rekognition이 대상 이미지에서 발견한 얼굴로서, 소스 이미지에서 분석한 얼굴과 일치하지 않는 얼굴 하나를 보여줍니다.


```
{
    "FaceMatches": [{
        "Face": {
            "BoundingBox": {
                "Width": 0.5521978139877319,
                "Top": 0.1203877404332161,
                "Left": 0.23626373708248138,
                "Height": 0.3126954436302185
            },
            "Confidence": 99.98751068115234,
            "Pose": {
                "Yaw": -82.36799621582031,
                "Roll": -62.13221740722656,
                "Pitch": 0.8652129173278809
            },
            "Quality": {
                "Sharpness": 99.99880981445312,
                "Brightness": 54.49755096435547
            },
            "Landmarks": [{
                    "Y": 0.2996366024017334,
                    "X": 0.41685718297958374,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.2658946216106415,
                    "X": 0.4414493441581726,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3465650677680969,
                    "X": 0.48636093735694885,
                    "Type": "nose"
                },
                {
                    "Y": 0.30935320258140564,
                    "X": 0.6251809000968933,
```

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

```
            },
            {
                "Y": 0.8081989884376526,
                "X": 0.48565614223480225,
                "Type": "mouthLeft"
            },
            {
                "Y": 0.7548204660415649,
                "X": 0.46090251207351685,
                "Type": "mouthRight"
            }
        ]
    }],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.5521978139877319,
            "Top": 0.1203877404332161,
            "Left": 0.23626373708248138,
            "Height": 0.3126954436302185
        },
        "Confidence": 99.98751068115234
    }
}
```


저장된 비디오에서 얼굴 감지

Amazon Rekognition Video는 Amazon S3 버킷에 저장된 비디오에서 얼굴을 감지하여 다음과 같은 정보를 제공할 수 있습니다.

- 비디오에서 얼굴이 감지된 시간
- 얼굴이 감지된 시점에 비디오 프레임에서 해당 얼굴의 위치
- 왼쪽 눈의 위치와 같은 얼굴 표식
- the section called “얼굴 속성에 대한 가이드라인” 페이지에 설명된 추가 속성.

Amazon Rekognition Video의 저장된 비디오 속 얼굴 감지는 비동기 작업입니다. 비디오에서 얼굴 인식을 시작하려면 StartFaceDetection을 직접 호출하세요. Amazon Rekognition Video는 동영상 분석 작업의 완료 상태를 Amazon Simple Notification Service(SNS) 주제에 게시합니다. 비디오 분석이 성공적으로 완료되면 GetFaceDetection을 직접 호출하여 비디오 분석 결과를 가져올 수 있습니다. 비디

오 분석 시작 및 결과 가져오기에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오.

이 절차는 동영상 분석 요청의 완료 상태를 가져오기 위해 Amazon Simple Queue Service(Amazon SQS) 대기열을 사용하는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)의 코드를 확장합니다.

Amazon S3 버킷에 저장된 비디오에서 얼굴을 감지하려면(SDK)

1. Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)을 수행합니다.
2. 1단계에서 만든 클래스 `VideoDetect`에 다음 코드를 추가합니다.

   AWS CLI

   - 다음 코드 샘플에서 `amzn-s3-demo-bucket` 및 `video-name`을 2단계에서 지정한 Amazon S3 버킷 이름과 파일 이름으로 변경합니다.
   - `region-name`을 사용 중인 AWS 리전으로 변경합니다. `profile_name`의 값을 개발자 프로필 이름으로 바꿉니다.
   - `TopicARN`을 Amazon Rekognition Video 구성의 3단계에서 생성한 Amazon SNS 주제의 ARN으로 변경합니다.
   - `RoleARN`을 Amazon Rekognition Video 구성의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 변경합니다.

   ```
   aws rekognition start-face-detection --video "{"S3Object":{"Bucket":"amzn-s3-
   demo-bucket","Name":"Video-Name"}}" --notification-channel
   "{"SNSTopicArn":"Topic-ARN","RoleArn":"Role-ARN"}" --region region-name --
   profile profile-name
   ```

   Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

   ```
   aws rekognition start-face-detection --video "{\"S3Object\":{\"Bucket\":\"amzn-
   s3-demo-bucket\",\"Name\":\"Video-Name\"}}" --notification-channel
   ```

```
"{\"SNSTopicArn\":\"Topic-ARN\",\"RoleArn\":\"Role-ARN\"}" --region region-name
 --profile profile-name
```

StartFaceDetection 작업을 실행하고 작업 ID 번호를 가져온 후 다음 GetFaceDetection 작업을 실행하고 작업 ID 번호를 제공합니다.

```
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);


}
```

main 함수에서 다음 줄을 바꿉니다.

```
        StartLabelDetection(amzn-s3-demo-bucket, video);
```

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

다음으로 바꿉니다.

```
        StartFaceDetection(amzn-s3-demo-bucket, video);

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

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//snippet-start:[rekognition.java2.recognize_video_faces.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_faces.import]


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

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

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
```

```
        "   bucket - The name of the bucket in which the video is located (for
example, (for example, amzn-s3-demo-bucket). \n\n"+
        "   video - The name of video (for example, people.mp4). \n\n" +
        "   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic. \n\n" +
        "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
role to use. \n\n" ;

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

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

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

    StartFaceDetection(rekClient, channel, bucket, video);
    GetFaceResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_faces.main]
public static void StartFaceDetection(RekognitionClient rekClient,
                                       NotificationChannel channel,
                                       String bucket,
                                       String video) {

    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
```

```
            .name(video)
            .build();

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

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

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

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

public static void GetFaceResults(RekognitionClient rekClient) {

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

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

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

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

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

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

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null
            VideoMetadata videoMetaData=faceDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

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

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

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

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

```
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_faces.main]
}
```

Python

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

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

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

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

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

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

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

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

main 함수에서 다음 줄을 바꿉니다.

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

다음으로 바꿉니다.

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

> **Note**
>
> Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 이외에 비디오 예제를 이미 실행한 경우, 바꿀 함수 이름이 다릅니다.

3. 코드를 실행합니다. 비디오에서 감지된 얼굴에 관한 정보가 표시됩니다.

GetFaceDetection 작업 응답

GetFaceDetection은 비디오에서 감지된 얼굴에 대한 정보가 포함된 배열(Faces)을 반환합니다. 배열 요소인 FaceDetection은 비디오에서 얼굴이 감지될 때마다 존재합니다. 반환된 배열 요소는 비디오 시작 후 시간별로(밀리초) 정렬되어 반환됩니다.

다음 예제는 GetFaceDetection의 부분 JSON 응답입니다. 응답에서 다음에 유의하십시오.

- 경계 상자 - 얼굴 주위를 두르는 경계 상자의 좌표.
- 신뢰도 - 경계 상자에 얼굴이 포함될 신뢰도 수준.
- 얼굴 표식 – 얼굴 표식의 배열. 응답은 왼쪽 눈, 오른쪽 눈, 입 같은 각각의 표식에 대해 x, y 좌표를 제공합니다.

- 얼굴 속성 - AgeRange, Beard, Emotions, Eyeglasses, EyesOpen, Gender, MouthOpen, Mustache, Smile, Sunglasses를 포함하는 얼굴 속성 집합입니다. 이 값은 부울(사람이 선글라스를 착용하고 있는지), 문자열(남성인지 여성인지) 등 다양한 형식이 될 수 있습니다. 또한 대부분의 속성의 경우, 응답은 해당 속성에 대해 감지된 값의 신뢰도도 제공합니다. 단, FaceOccluded 속성과 EyeDirection 속성은 `DetectFaces`를 사용할 때는 지원되지만, `StartFaceDetection` 및 `GetFaceDetection`을 사용하여 비디오를 분석할 때는 지원되지 않습니다.
- 타임스탬프 - 비디오에서 얼굴이 감지된 시간입니다.
- 페이징 정보 - 이 예제는 얼굴 감지 정보의 페이지 하나를 보여줍니다. `GetFaceDetection`의 `MaxResults` 입력 파라미터에 반환될 사람 요소의 수를 지정할 수 있습니다. `MaxResults` 보다 많은 결과가 존재할 경우 `GetFaceDetection`은 결과의 다음 페이지를 가져올 때 사용되는 토큰(`NextToken`)을 반환합니다. 자세한 내용은 Amazon Rekognition Video 분석 결과 가져오기 단원을 참조하십시오.
- 비디오 정보 - 이 응답은 `GetFaceDetection`이 반환하는 정보의 각 페이지에 있는 비디오 형식(`VideoMetadata`)에 관한 정보를 포함합니다.
- 품질 – 얼굴의 밝기와 선명도를 기술합니다.
- 자세 – 이미지 내 얼굴의 회전을 기술합니다.


```
{
    "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
    }
}
```

컬렉션에서 얼굴 검색

Amazon Rekognition을 사용하면 입력된 얼굴을 사용하여 저장된 얼굴 컬렉션에서 일치하는 얼굴을 검색할 수 있습니다. 먼저 탐지된 얼굴에 대한 정보를 "컬렉션"이라는 서버 측 컨테이너에 저장합니다. 컬렉션에는 개별 얼굴과 사용자(동일한 사람의 여러 얼굴)가 모두 저장됩니다. 개별 얼굴은 (얼굴의 실제 이미지가 아닌) 얼굴을 수학적으로 표현한 얼굴 벡터로 저장됩니다. 동일한 사람의 다양한 이미지를 사용하여 동일한 컬렉션에 여러 개의 얼굴 벡터를 만들고 저장할 수 있습니다. 그런 다음 동일한 사람의 여러 얼굴 벡터를 집계하여 사용자 벡터를 만들 수 있습니다. 사용자 벡터는 여러 정도의 조명, 선명도, 포즈, 외양 등을 포함하는 보다 강력한 묘사를 통해 더 정확한 얼굴 검색을 제공할 수 있습니다.

컬렉션을 만든 후에는 입력된 얼굴을 사용하여 컬렉션에서 일치하는 사용자 벡터 또는 얼굴 벡터를 검색할 수 있습니다. 사용자 벡터를 기준으로 검색하면 개별 얼굴 벡터로 검색하는 것보다 정확도를 크게 향상시킬 수 있습니다. 이미지, 저장된 비디오 및 스트리밍 비디오에서 감지된 얼굴을 사용하여 저장된 얼굴 벡터를 검색할 수 있습니다. 이미지에서 감지된 얼굴을 사용하여 저장된 사용자 벡터를 검색할 수 있습니다.

얼굴 정보를 저장하려면 다음 과정을 수행해야 합니다.

1. 컬렉션 생성 - 얼굴 정보를 저장하려면 먼저 계정의 AWS 리전 중 하나에 얼굴 컬렉션을 생성(CreateCollection)해야 합니다. `IndexFaces` 작업을 호출할 때 이 얼굴 모음을 지정합니다.
2. 얼굴 인덱싱 - IndexFaces 작업은 이미지에서 얼굴을 감지하고 얼굴 벡터를 추출하여 컬렉션에 저장합니다. 이 작업을 사용하여 이미지에서 얼굴을 감지하고, 감지된 얼굴 특징에 대한 정보를 모음으로 유지할 수 있습니다. 이것이 스토리지 기반 API 작업의 예인 이유는 서비스가 얼굴 벡터 정보를 서버에 저장하기 때문입니다.

사용자를 생성하고 여러 개의 얼굴 벡터를 사용자와 연결하려면 다음 작업을 수행해야 합니다.

1. 사용자 생성 - 먼저 CreateUser를 사용하여 사용자를 생성해야 합니다. 같은 인물의 여러 얼굴 벡터를 하나의 사용자 벡터로 집계하여 얼굴 일치의 정확도를 높일 수 있습니다. 최대 100개의 얼굴 벡터를 하나의 사용자 벡터와 연결할 수 있습니다.
2. 얼굴 연결 - 사용자를 생성한 후 AssociateFaces 작업을 통해 기존 얼굴 벡터를 해당 사용자에 추가할 수 있습니다. 얼굴 벡터가 사용자 벡터와 연결되려면 해당 사용자 벡터와 같은 컬렉션에 있어야 합니다.

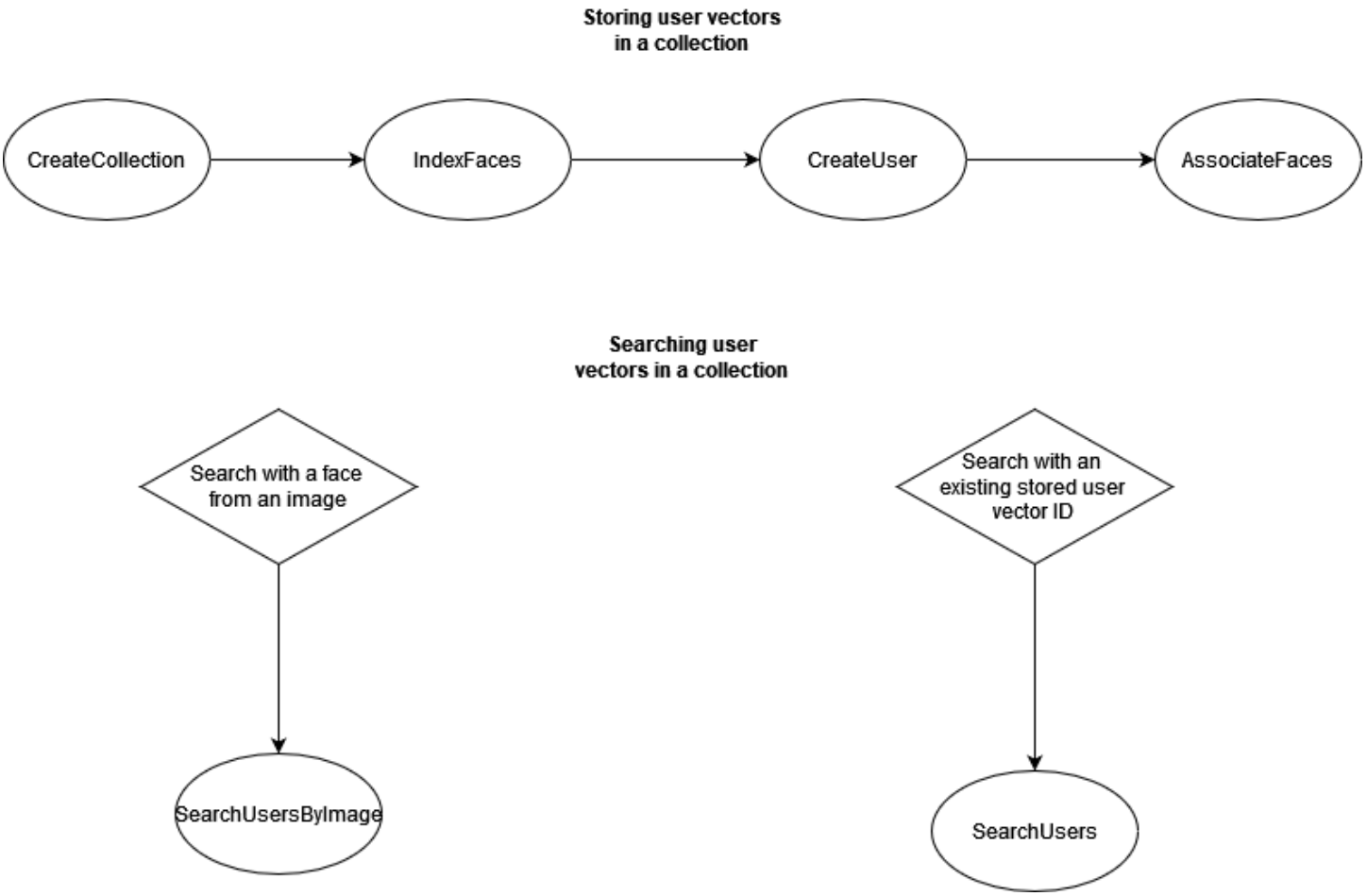
컬렉션을 생성하고 얼굴 및 사용자 벡터를 저장한 후에는 다음 작업을 사용하여 일치하는 얼굴을 검색할 수 있습니다.

- SearchFacesByImage - 이미지에 있는 얼굴로 저장된 개별 얼굴을 검색합니다.
- SearchFaces - 제공된 얼굴 ID로 저장된 개별 얼굴을 검색합니다.
- SearchUsers - 제공된 얼굴 ID 또는 사용자 ID를 사용하여 저장된 사용자를 검색합니다.
- SearchUsersByImage - 이미지에 있는 얼굴로 저장된 사용자를 검색합니다.
- StartFaceSearch - 저장된 비디오에서 얼굴을 검색합니다.
- CreateStreamProcessor - 스트리밍 비디오에서 얼굴을 검색합니다.

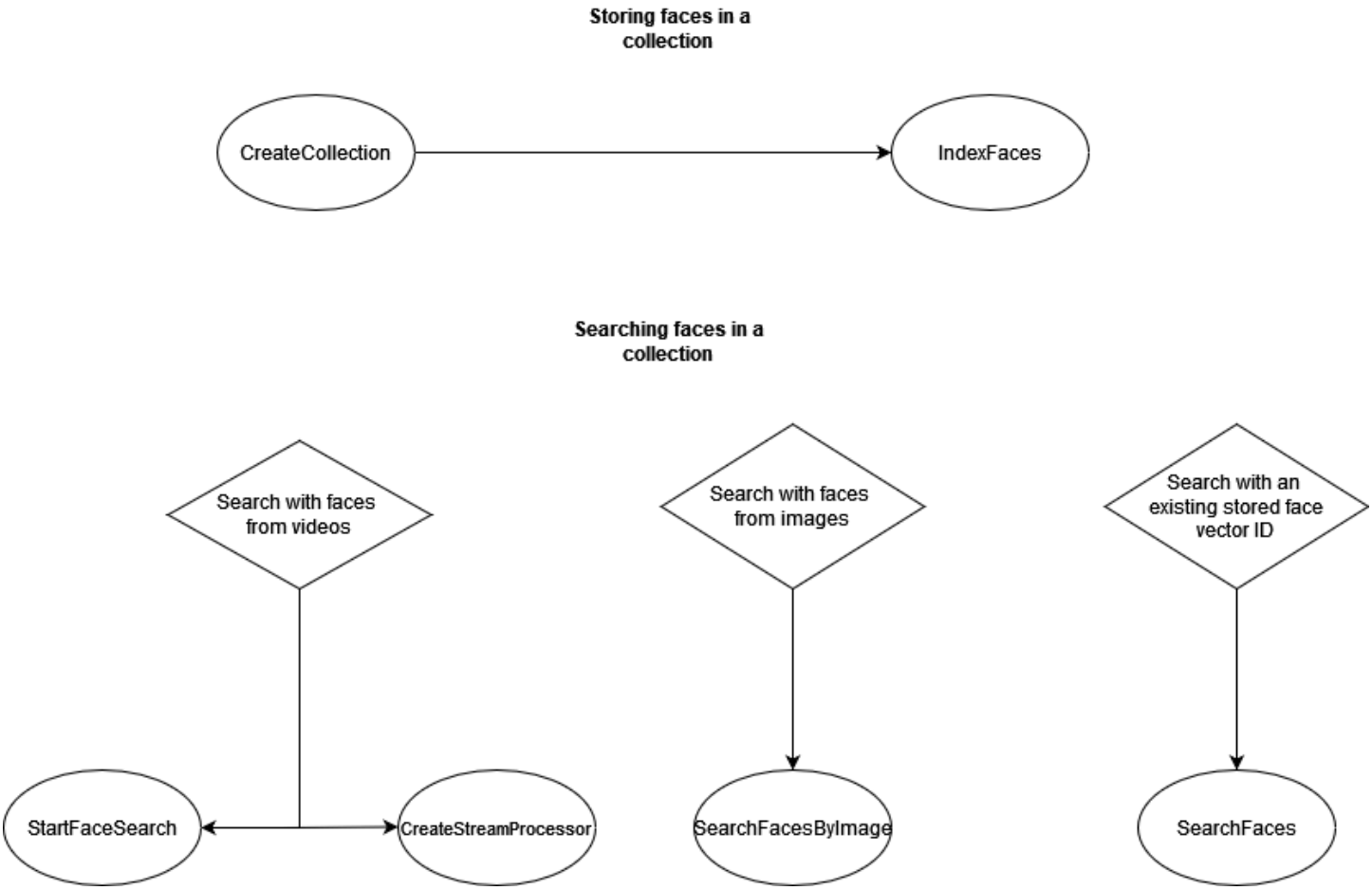
> **Note**
>
> 컬렉션에는 얼굴의 수학적 표현인 얼굴 벡터가 저장됩니다. 컬렉션에는 얼굴 이미지가 저장되지 않습니다.

다음 다이어그램은 컬렉션 사용 목표에 따른 직접 호출 작업의 순서를 보여줍니다.

사용자 벡터와 최대 정확도로 매칭:



개별 얼굴 벡터와 높은 정확도로 매칭:



여러 가지 일반적인 사용자 시나리오를 위한 자습서 모음. 예를 들어, `IndexFaces` 및 `AssociateFaces` 작업을 사용하여 스캔한 직원 배지 이미지와 정부에서 발급한 ID에서 탐지된 얼굴을 저장하는 얼굴 컬렉션을 만들 수 있습니다. 직원이 건물에 들어오면 직원 얼굴 이미지가 캡처되어 `SearchUsersByImage` 작업으로 전송됩니다. 얼굴 일치에서 충분히 높은 유사성 점수(가령 99%)가 나오면 직원을 인증할 수 있습니다.

얼굴 컬렉션, 얼굴 및 사용자 관리

얼굴 컬렉션은 Amazon Rekognition에서 얼굴 및 사용자와 관련된 정보를 포함하는 데 사용하는 리소스입니다. 다양한 Rekognition API 작업을 사용하여 컬렉션 자체와 컬렉션에 저장된 얼굴 또는 얼굴 벡터를 관리합니다.

컬렉션 관리

얼굴 컬렉션은 기본 Amazon Rekognition 리소스이며, 생성되는 각각의 얼굴 컬렉션에는 고유의 Amazon 리소스 이름(ARN)이 있습니다. 계정의 특정 AWS 리전에서 각 얼굴 모음을 생성합니다. 모음

을 만들면, 최신 버전의 얼굴 감지 모델에 연결됩니다. 자세한 내용은 모델 버전 관리 이해 단원을 참조하십시오.

컬렉션에서 다음 관리 작업을 수행할 수 있습니다.

- CreateCollection을 사용하여 컬렉션을 생성합니다. 자세한 내용은 컬렉션 생성 단원을 참조하십시오.
- ListCollections를 사용하여 사용 가능한 컬렉션을 조회합니다. 자세한 내용은 컬렉션 나열 단원을 참조하십시오.
- DescribeCollection으로 컬렉션을 설명합니다. 자세한 내용은 컬렉션 설명 단원을 참조하십시오.
- DeleteCollection을 사용하여 컬렉션을 삭제합니다. 자세한 내용은 컬렉션 삭제 단원을 참조하십시오.

컬렉션에서 얼굴 관리

얼굴 컬렉션을 만든 후에는 그 안에 얼굴을 저장할 수 있습니다. Amazon Rekognition은 컬렉션의 얼굴 관리를 위해 다음 작업을 제공합니다.

- IndexFaces 작업은 입력 이미지(JPEG 또는 PNG)에서 얼굴을 감지하여 지정된 얼굴 컬렉션에 추가합니다. 이미지에서 감지된 각 얼굴에 대해 고유한 얼굴 ID가 반환됩니다. 얼굴을 유지한 후 얼굴 모음에서 얼굴 일치를 검색할 수 있습니다. 자세한 내용은 컬렉션에 얼굴 추가 단원을 참조하십시오.
- ListFaces 작업은 컬렉션에 있는 얼굴을 나열합니다. 자세한 내용은 컬렉션에 얼굴 추가 단원을 참조하십시오.
- DeleteFaces 작업은 컬렉션에서 얼굴을 삭제합니다. 자세한 내용은 컬렉션에서 얼굴 삭제 단원을 참조하십시오.

컬렉션의 사용자 관리

같은 사람의 얼굴 벡터를 여러 개 저장한 후에는 이러한 모든 얼굴 벡터를 하나의 사용자 벡터로 연결하여 정확도를 높일 수 있습니다. 다음 작업을 통해 사용자를 관리할 수 있습니다.

- CreateUser - 이 작업은 제공된 고유한 사용자 ID를 사용하여 컬렉션에 새 사용자를 생성합니다.
- AssociateUsers - 하나의 사용자 ID에 1~100개의 고유한 얼굴 ID를 추가합니다. 최소 한 개의 얼굴 ID를 사용자와 연결한 후에는 컬렉션에서 해당 사용자와 일치하는 항목을 검색할 수 있습니다.
- ListUsers - 컬렉션에 있는 사용자를 나열합니다.

- DeleteUsers - 제공된 사용자 ID를 이용하여 컬렉션에서 사용자를 삭제합니다.
- DisassociateFaces - 사용자로부터 하나 이상의 얼굴 ID를 제거합니다.

얼굴 연결 및 매칭을 위한 유사성 임곗값 사용

유사성 임곗값은 얼굴을 연결하고 일치시키는 데 사용됩니다. 다음은 두 사용 사례에 대해 유사성 임곗값을 사용하기 위한 지침입니다.

얼굴 연결을 위한 유사성 임계값 사용

AssociateFaces 작업을 사용하여 얼굴을 연결할 때 사용자와 연결된 얼굴이 모두 동일한 사람인지 확인하는 것이 중요합니다. UserMatchThreshold 파라미터는 이를 돕기 위해 새 얼굴이 최소 하나 이상의 FaceID를 이미 포함하는 UserID와 연결될 때 필요한 최소 사용자 일치 신뢰도를 지정합니다. 이렇게 하면 FaceIds가 올바른 UserID와 연결되도록 할 수 있습니다. 이 값의 범위는 0~100이고 기본값은 75입니다.

유사성 임계값을 사용하여 얼굴 일치

유사성 임계값을 입력 파라미터로 제공하여 모든 검색 작업(CompareFaces, SearchFaces, SearchFacesByImage, SearchUsers, SearchUsersByImage)의 결과를 제어할 수 있습니다.

FaceMatchThreshold는 SearchFaces 및 SearchFacesByImage에 대한 유사성 임계값 입력 속성이며 일치를 수행 중인 얼굴의 유사성에 따라 반환되는 결과의 수를 제어합니다. SearchUsers 및 SearchUsersByImage에 대한 유사성 임계값 입력 속성은 UserMatchThreshold이며 일치를 수행 중인 사용자 벡터의 유사성에 따라 반환되는 결과의 수를 제어합니다. CompareFaces의 임계값 속성은 SimilarityThreshold입니다.

Similarity 응답 속성 값이 임계값보다 낮은 응답은 반환되지 않습니다. 이 임계값은 일치 결과에 포함된 가양성의 수를 결정할 수 있기 때문에 사용 사례에 맞게 보정하기 위해 중요합니다. 이 값은 검색 결과의 재현율을 제어합니다. 임계값이 낮을수록 재현율은 높아집니다.

모든 기계 학습 시스템은 확률론적입니다. 사용 사례에 따라 적절한 유사성 임계값을 설정할 때에는 본인의 판단을 활용해야 합니다. 예를 들어 비슷하게 생긴 가족을 식별하는 포토 앱을 만들려는 경우 임계값을 낮게(예:80%) 선택할 수 있습니다. 반면, 많은 법 집행 사용 사례의 경우, 실수로 잘못 식별되는 경우를 줄이기 위해 99% 이상의 높은 임계값을 사용하는 것이 좋습니다.

FaceMatchThreshold와 UserMatchThreshold 외에, 실수로 잘못 식별되는 경우를 줄이기 위한 수단으로 Similarity 응답 속성을 사용할 수 있습니다. 예를 들어 낮은 임계값(80%)을 사용하기로

선택하여 더 많은 결과를 반환할 수 있습니다. 그런 다음 유사성 응답 속성(유사성의 정도 %)를 사용하여 선택 범위를 좁히고 해당 애플리케이션에서 올바른 응답을 필터링할 수 있습니다. 다시 유사성을 높이면(예: 99% 이상) 잘못된 식별이 줄어들 것입니다.

일반적인 시나리오에서 얼굴 인덱싱 지침

다음은 일반적인 시나리오에서 IndexFaces를 사용하기 위한 지침입니다.

중요 또는 공공 안전 애플리케이션

- 각 이미지에 얼굴이 하나씩만 포함된 이미지를 사용하여 IndexFaces를 직접 호출하고 반환된 얼굴 ID를 이미지 내 인물의 식별자와 연결합니다.
- 인덱싱 전에 DetectFaces를 사용하여 이미지에 얼굴이 하나뿐인지 확인할 수 있습니다. 얼굴이 여러 개 감지된 경우 검토 후 얼굴이 하나만 있는 이미지를 다시 제출합니다. 이는 잘못하여 복수의 얼굴을 인덱싱하고 같은 사람에 연결하는 것을 방지합니다.

사진 공유 및 소셜 미디어 애플리케이션

- 가족 앨범과 같은 사용 사례에서는 여러 얼굴이 포함된 이미지에 대한 제한 없이 IndexFaces를 호출해야 합니다. 이런 경우, 모든 사진에서 각 개인을 식별하고 해당 정보를 사용하여 사진 안의 사람들에 따라 사진을 그룹화해야 합니다.

일반 사용 사례

- 같은 사람의 여러 이미지, 특히 다양한 얼굴 속성(얼굴 포즈, 얼굴의 털 등)을 가진 이미지를 인덱싱하고 사용자를 생성하여 해당 사용자와 다양한 얼굴을 연결함으로써 일치 품질을 개선하세요.
- 실패한 일치를 올바른 얼굴 식별자로 인덱싱하여 이후의 얼굴 일치 기능을 개선할 수 있도록 검토 프로세스를 포함시킵니다.
- 이미지 품질에 대한 자세한 내용은 얼굴 비교 입력 이미지에 대한 권장 사항 단원을 참조하십시오.

컬렉션 내의 얼굴 및 사용자 검색

얼굴 컬렉션을 만들고 얼굴 벡터 및/또는 사용자 벡터를 저장한 후에는 얼굴 컬렉션에서 얼굴 일치를 검색할 수 있습니다. Amazon Rekognition에서는 다음과 일치하는 얼굴을 컬렉션에서 검색할 수 있습니다.

- 제공된 얼굴 ID(SearchFaces). 자세한 내용은 얼굴 ID로 얼굴 검색 단원을 참조하십시오.
- 제공된 이미지 내에서 가장 큰 얼굴(SearchFacesByImage). 자세한 내용은 이미지를 사용하여 얼굴 검색 단원을 참조하십시오.
- 저장된 비디오의 얼굴. 자세한 내용은 저장된 비디오에서 얼굴 검색 단원을 참조하십시오.
- 스트리밍 비디오의 얼굴. 자세한 내용은 스트리밍 비디오 이벤트 작업 단원을 참조하십시오.

CompareFaces 작업을 사용해 원본 이미지의 얼굴과 대상 이미지의 얼굴을 비교할 수 있습니다. 이 비교의 범위는 대상 이미지에서 감지된 얼굴로 제한됩니다. 자세한 내용은 이미지에 있는 얼굴 비교를 참조하세요.

다음 목록에 나와 있는 다양한 검색 작업은 특정 얼굴(FaceId 또는 입력 이미지로 식별됨)을 지정된 얼굴 컬렉션에 저장된 모든 얼굴과 비교합니다.

- SearchFaces
- SearchFacesByImage
- SearchUsers
- SearchUsersByImage

공공 안전 관련 사용 사례

공공 안전과 관련된 사용 사례에서 얼굴 감지 및 비교 시스템을 배포하는 경우 센서, 입력 이미지 및 비디오 모범 사례 및 일반적인 시나리오에서 얼굴 인덱싱 지침에 나열된 권장 사항 이외에 다음 모범 사례를 따라야 합니다. 첫째, 오류 및 오탐을 줄이기 위해 99% 이상의 신뢰도 임계값을 사용해야 합니다. 둘째, 인간 검토자가 얼굴 감지 또는 비교 시스템이 제공한 결과를 확인해야 하며, 인간에 의한 추가 검토 없이 시스템 출력에 기반하여 의사 결정을 하면 안 됩니다. 얼굴 감지 및 비교 시스템은 검토 범위를 좁혀 인간 작업자가 신속하게 검토하고 옵션을 고려할 수 있게 해주는 도구로 사용해야 합니다. 셋째, 가능할 경우 최종 사용자 및 피사 인물에게 이러한 시스템의 사용을 고지하고, 사용에 대한 동의를 얻고, 최종 사용자 및 피사 인물이 시스템 개선을 위한 피드백을 제공할 수 있는 메커니즘을 제공하는 등 이러한 사용 사례에서 얼굴 감지 및 비교 시스템을 투명하게 사용해야 합니다.

귀하가 범죄 수사와 관련하여 Amazon Rekognition 얼굴 비교 기능을 사용하는 법 집행 기관인 경우 AWS 서비스 약관에 명시된 요구 사항을 준수해야 합니다. 다음 내용이 해당됩니다.

- 적절한 교육을 받은 사람이 개인의 시민적 자유 또는 동등한 인권에 영향을 줄 수 있는 조치를 취하기 위해 모든 결정을 검토합니다.
- 얼굴 인식 시스템의 책임 있는 사용에 대해 직원을 교육합니다.

- 얼굴 인식 시스템 사용에 대한 공개 정보를 제공합니다.
- 독립적인 검토가 없거나 급박한 상황이 아닌 경우 개인의 지속적인 감시에 Amazon Rekognition을 사용해서는 안 됩니다.

얼굴 비교 일치 결과는 항상 다른 결정적인 증거와 함께 고려되어야 하며 조치를 취할 때 유일한 결정 요인으로 사용되어서는 안 됩니다. 하지만 얼굴 비교가 법 집행과 무관한 상황에서 사용될 경우(예: 전화를 잠금 해제하거나 보안 시스템이 적용된 개인 건물에 액세스하기 위해 피고용인의 ID를 인증하는 경우), 개인의 시민적 자유 또는 동등한 인권에 영향을 미치지 않으므로 이러한 결정을 내릴 때 수동 감사가 필요하지 않습니다.

공공 안전과 관련된 사용 사례에서 얼굴 감지 또는 얼굴 비교 시스템을 사용하려는 경우 앞서 언급한 모범 사례를 사용해야 합니다. 또한 얼굴 비교 사용에 대해 발표된 리소스를 참고해야 합니다. 여기에는 미 법무부 법무지원국에서 제공하는 Face Recognition Policy Development Template For Use In Criminal Intelligence and Investigative Activities(범죄 정보 및 수사 활동에 활용하기 위한 얼굴 인식 정책 개발 템플릿)가 포함됩니다. 이 템플릿은 다양한 얼굴 비교 및 생체 인식 관련 리소스를 제공하며, 법 집행 및 공공 안전 기관에 관계법을 준수하고 프라이버시 침해 위험을 낮추며 책임 및 감독을 확립하는 얼굴 비교 정책을 개발하기 위한 프레임워크를 제공하도록 고안되었습니다. 추가 리소스로는 미 통신정보관리청의 Best Privacy Practices for Commercial Use of Facial Recognition(얼굴 인식의 상업적 활용을 위한 개인정보보호 모범 사례)과 미 연방통상위원회의 Best Practices for Common Uses of Facial Recognition(얼굴 인식의 일반적 사용을 위한 모범 사례)가 포함됩니다. 향후 다른 리소스가 개발 및 공개될 수 있으므로 이 중요한 주제에 지속적으로 관심을 기울여야 합니다.

AWS 서비스를 사용하는 고객은 모든 관계법을 준수해야 하며, AWS 서비스를 다른 사람의 권리를 침해하거나 다른 사람에게 피해를 줄 수 있는 방식으로 사용할 수 없습니다. 이는 공공 안전을 위한 AWS 서비스를 불법적으로 특정 개인을 차별하거나 개인의 정당한 프로세스, 프라이버시 또는 시민적 자유를 침해하는 방식으로 사용할 수 없다는 의미입니다. 필요에 따라 적절한 법률 조언을 받아 사용 사례에 대한 법적 요건 등을 검토해야 합니다.

공공 안전을 위한 Amazon Rekognition 사용

Amazon Rekognition은 미아 찾기, 인신매매 단속 또는 범죄 예방과 같은 공공 안전 및 법 집행 시나리오에서 도움이 될 수 있습니다. 공공 안전 및 법 집행 시나리오에서 다음을 고려하십시오.

- 가능성 있는 일치 인물을 찾기 위한 첫 단계로 Amazon Rekognition을 사용합니다. Amazon Rekognition 얼굴 작업의 응답을 통해 향후 고려할 잠재적인 일치 세트를 빠르게 얻을 수 있습니다.
- 인간의 분석을 필요로 하는 시나리오에서 자율적 결정을 내리는 데 Amazon Rekognition 응답을 사용하지 마십시오. 귀하가 범죄 수사와 관련하여 개인을 식별하는 데 도움을 받기 위해 Amazon Rekognition을 사용하는 법 집행 기관이며, 식별 결과를 근거로 해당 개인의 시민적 자유 또는 그에

동등한 인권에 영향을 줄 수 있는 조치를 취할 경우, 해당 조치의 결정은 적절한 훈련을 받은 사람이 독립적으로 조사한 신원 증빙의 근거에 기반하여 내려야 합니다.

- 매우 정확한 얼굴 유사성 일치가 필요한 시나리오의 경우 99% 유사성 임계값을 사용합니다. 빌딩 액세스 인증을 예로 들 수 있습니다.
- 법 집행을 위한 사용 시나리오 등 시민의 권리를 고려해야 할 경우 99% 이상의 신뢰도 임계값을 사용하고 개인의 권리를 침해하지 않도록 얼굴 비교 예측 내용에 대한 인간의 검토를 도입해야 합니다.
- 대규모의 잠재적 일치 세트를 통해 이점을 얻게 되는 시나리오의 경우 99% 미만의 유사성 임계값을 사용합니다. 실종된 사람을 찾는 것을 예로 들 수 있습니다. 필요한 경우 유사성 응답 속성을 사용하여 잠재적 일치가 인식하고자 하는 사람과 어느 정도로 유사한지 판단할 수 있습니다.
- Amazon Rekognition에서 반환하는 가양성 얼굴 일치에 대한 계획을 마련하세요. 예를 들어 IndexFaces 작업을 사용하여 인덱스를 구축할 때 동일한 사람의 이미지를 여러 개 사용하여 일치를 개선합니다. 자세한 내용은 일반적인 시나리오에서 얼굴 인덱싱 지침 단원을 참조하십시오.

다른 사용 사례에서는(예: 소셜 미디어), 최선의 판단을 통해 Amazon Rekognition 결과에 인간의 검토가 필요한지 여부를 평가하는 것이 좋습니다. 또한 애플리케이션 요구 사항에 따라 유사성 임계값이 낮아질 수 있습니다.

컬렉션 생성

CreateCollection 작업을 사용하여 컬렉션을 생성할 수 있습니다.

자세한 내용은 컬렉션 관리 단원을 참조하십시오.

모음을 만들려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. `AmazonRekognitionFullAccess` 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 `CreateCollection` 작업을 호출합니다.

   Java

   다음 예제는 모음을 만들고 Amazon Resource Name(ARN)을 표시합니다.

   `collectionId`의 값을, 만들려는 모음의 이름으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
//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

이 AWS CLI 명령은 `create-collection` CLI 작업에 대한 JSON 출력을 표시합니다.

`collection-id`의 값을, 만들려는 모음의 이름으로 바꿉니다.

profile_name의 값을 개발자 프로필 이름으로 바꿉니다.

```
aws rekognition create-collection --profile profile-name --collection-id
 "collection-name"
```

Python

다음 예제는 모음을 만들고 Amazon Resource Name(ARN)을 표시합니다.

collection_id의 값을, 만들려는 모음의 이름으로 변경합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

다음 예제는 모음을 만들고 Amazon Resource Name(ARN)을 표시합니다.

collectionId의 값을, 만들려는 모음의 이름으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

다음 예시에서는 `region` 값을 계정과 연결된 리전 이름으로 바꾸고 `collectionName` 값을 원하는 컬렉션 이름으로 바꿉니다.

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

`CreationCollection`에 대한 입력은 생성하려는 모음의 이름입니다.

```
{
```

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

CreateCollection 작업 응답

Amazon Rekognition은 컬렉션을 만들고 새로 만든 컬렉션의 Amazon 리소스 이름(ARN)을 반환합니다.

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

컬렉션 태그 지정

태그를 사용하여 Amazon Rekognition 컬렉션을 식별, 구성, 검색 및 필터링할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다.

Identity and Access Management(IAM)을 통해 태그를 사용하여 컬렉션에 대한 액세스를 제어할 수도 있습니다. 자세한 내용은 AWS 리소스 태그를 사용하여 리소스에 대한 액세스 제어를 참조하세요.

주제

- 새 컬렉션에 태그 추가
- 기존 컬렉션에 태그 추가
- 컬렉션의 태그 나열
- 컬렉션에서 태그 삭제

새 컬렉션에 태그 추가

`CreateCollection` 작업을 사용하여 컬렉션을 생성할 때 태그를 추가할 수 있습니다. `Tags` 배열 입력 파라미터에 하나 이상의 태그를 지정합니다.

AWS CLI

`profile_name`의 값을 개발자 프로필 이름으로 바꿉니다.

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

Windows 디바이스의 경우는 다음과 같이 하십시오.

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

Python

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
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()
```

기존 컬렉션에 태그 추가

기존 컬렉션에 하나 이상의 태그를 추가하려면 `TagResource` 작업을 사용합니다. 컬렉션의 Amazon 리소스 이름(ARN)(`ResourceArn`)과 추가할 태그(`Tags`)를 지정합니다. 다음 예제는 태그 2개를 추가하는 방법을 보여줍니다.

AWS CLI

`profile_name`의 값을 개발자 프로필 이름으로 바꿉니다.

```
aws rekognition tag-resource --resource-arn collection-arn --tags
 "{"key1":"value1","key2":"value2"}" --profile profile-name
```

Windows 디바이스의 경우는 다음과 같이 하십시오.

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

Python

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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**
>
> 컬렉션의 Amazon 리소스 이름을 모르는 경우 `DescribeCollection` 작업을 사용할 수 있습니다.

컬렉션의 태그 나열

컬렉션에 연결된 태그를 나열하려면 `ListTagsForResource` 작업을 사용하고 컬렉션의 ARN(`ResourceArn`)을 명시하세요. 응답은 지정된 컬렉션에 연결된 태그 키 및 값의 맵입니다.

AWS CLI

`profile_name`의 값을 개발자 프로필 이름으로 바꿉니다.

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

Python

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
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()
```

출력된 결과에 컬렉션에 연결된 태그 목록이 표시됩니다.

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

컬렉션에서 태그 삭제

컬렉션에서 하나 이상의 태그를 삭제하려면 UntagResource 작업을 사용합니다. 제거하려는 모델의 ARN(ResourceArn)과 태그 키(Tag-Keys)를 지정합니다.

AWS CLI

profile_name의 값을 개발자 프로필 이름으로 바꿉니다.

```
aws rekognition untag-resource --resource-arn resource-arn --profile profile-name --
tag-keys "key1" "key2"
```

또는 다음 형식으로 tag-key를 지정할 수도 있습니다.

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

Python

Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
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()
```

컬렉션 나열

ListCollections 작업을 사용하여 현재 사용하는 리전의 컬렉션을 나열할 수 있습니다.

자세한 내용은 컬렉션 관리 단원을 참조하십시오.

모음을 나열하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.
   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.
   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제를 사용하여 ListCollections 작업을 호출합니다.

   Java

   다음 예제는 현재 리전의 모음을 나열합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//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

이 AWS CLI 명령은 `list-collections` CLI 작업에 대한 JSON 출력을 표시합니다. `profile_name`의 값을 개발자 프로필 이름으로 바꿉니다.

```
aws rekognition list-collections --profile profile-name
```

Python

다음 예제는 현재 리전의 모음을 나열합니다.

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
#Copyright 2018 Amazon.com, Inc. or its 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

다음 예제는 현재 리전의 모음을 나열합니다.

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

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

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

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

        ListCollectionsResponse listCollectionsResponse = null;
```

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

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

            listCollectionsResponse =
 rekognitionClient.ListCollections(listCollectionsRequest);

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

Node.js

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

`ListCollections`에 대한 입력은 반환될 최대 모음 수입니다.

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

응답의 컬렉션이 `MaxResults`가 요청한 것보다 많으면 후속 `ListCollections` 직접 호출에서 다음 결과 집합을 가져오는 데 사용할 수 있는 토큰이 반환됩니다. 예제:

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

ListCollections 작업 응답

Amazon Rekognition은 컬렉션의 배열(CollectionIds)을 반환합니다. 각 모음의 얼굴을 분석하는 데 사용된 얼굴 모델 버전은 별도의 배열(FaceModelVersions)로 제공됩니다. 예를 들어 다음 JSON 응답에서 MyCollection 모음은 얼굴 모델 버전 2.0을 사용하여 얼굴을 분석합니다. AnotherCollection모음은 얼굴 모델 버전 3.0을 사용합니다. 자세한 내용은 모델 버전 관리 이해 단원을 참조하십시오.

NextToken은 ListCollections에 대한 후속 호출에서 다음 결과 집합을 가져오는 데 사용되는 토큰입니다.

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

컬렉션 설명

DescribeCollection 작업을 사용하여 컬렉션에 관한 다음 정보를 얻을 수 있습니다.

- 모음에 인덱싱된 얼굴의 수.
- 컬렉션과 함께 사용되는 모델 버전. 자세한 내용은 the section called “모델 버전 관리 이해” 단원을 참조하십시오.
- 모음의 Amazon 리소스 이름(ARN).
- 모음의 생성 날짜 및 시간.

모음을 설명하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

    a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 DescribeCollection 작업을 호출합니다.

Java

이 예제는 모음을 설명합니다.

collectionId의 값을 원하는 모음의 ID로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
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

이 AWS CLI 명령은 `describe-collection` CLI 작업에 대한 JSON 출력을 표시합니다. `collection-id`의 값을 원하는 모음의 ID로 변경합니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
aws rekognition describe-collection --collection-id collection-name --profile
 profile-name
```

Python

이 예제는 모음을 설명합니다.

`collection_id`의 값을 원하는 모음의 ID로 변경합니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 모음을 설명합니다.

`collectionId`의 값을 원하는 모음의 ID로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

`DescribeCollection`에 대한 입력은 원하는 모음의 ID입니다. 다음 JSON 예제를 참조하십시오.

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

DescribeCollection 작업 응답

응답에는 다음이 포함됩니다.

- 모음에 인덱싱된 얼굴의 수, `FaceCount`.
- 컬렉션과 함께 사용되는 얼굴 모델 버전, `FaceModelVersion`. 자세한 내용은 the section called “모델 버전 관리 이해” 단원을 참조하십시오.
- 모음의 Amazon 리소스 이름, `CollectionARN`.
- 모음의 생성 날짜 및 시간, `CreationTimestamp`. `CreationTimestamp`의 값은 밀리초인데 모음 생성까지의 Unix epoch 시간 때문입니다. Unix epoch 시간은 1970년 1월 1일, 목요일 00:00:00 협정 세계시(UTC)입니다. 자세한 내용은 Unix Time을 참조하십시오.

```
{
    "CollectionARN": "arn:aws:rekognition:us-east-1:nnnnnnnnnnnn:collection/
MyCollection",
    "CreationTimestamp": 1.533422155042E9,
```

```
    "FaceCount": 200,
    "UserCount" : 20,
    "FaceModelVersion": "1.0"
}
```

컬렉션 삭제

DeleteCollection 작업을 사용하여 컬렉션을 삭제할 수 있습니다.

자세한 내용은 컬렉션 관리 단원을 참조하십시오.

모음을 삭제하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.
   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.
   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제를 사용하여 DeleteCollection 작업을 호출합니다.

   Java

   이 예제는 모음을 삭제합니다.

   collectionId 값을, 삭제하려는 모음으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
// 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

이 AWS CLI 명령은 delete-collection CLI 작업에 대한 JSON 출력을 표시합니다. collection-id의 값을, 삭제하려는 모음의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
aws rekognition delete-collection --collection-id collection-name --profile
 profile-name
```

Python

이 예제는 모음을 삭제합니다.

collection_id 값을, 삭제하려는 모음으로 변경합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 모음을 삭제합니다.

`collectionId` 값을, 삭제하려는 모음으로 변경합니다.

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

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

public class DeleteCollection
{
```

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

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

        DeleteCollectionRequest deleteCollectionRequest = new
 DeleteCollectionRequest()
        {
            CollectionId = collectionId
        };

        DeleteCollectionResponse deleteCollectionResponse =
 rekognitionClient.DeleteCollection(deleteCollectionRequest);
        Console.WriteLine(collectionId + ": " +
 deleteCollectionResponse.StatusCode);
    }
}
```

Node.js

Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

DeleteCollection에 대한 입력은 삭제할 모음의 ID입니다. 다음 JSON 예제를 참조하십시오.

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

DeleteCollection 작업 응답

DeleteCollection 응답에는 작업의 성공 또는 실패를 나타내는 HTTP 상태 코드가 들어 있습니다. 모음이 성공적으로 삭제되면 200이 반환됩니다.

```
{"StatusCode":200}
```

컬렉션에 얼굴 추가

IndexFaces 작업을 사용하여 이미지에서 얼굴을 감지하고 컬렉션에 추가할 수 있습니다. Amazon Rekognition은 감지된 각 얼굴에서 얼굴 특성을 추출하여 데이터베이스에 이 특성 정보를 저장합

니다. 또한 이 명령은 감지된 각 얼굴의 메타데이터를 지정된 얼굴 컬렉션에 저장합니다. Amazon Rekognition은 실제 이미지 바이트를 저장하지 않습니다.

인덱싱에 적합한 얼굴 제공에 대한 자세한 내용은 얼굴 비교 입력 이미지에 대한 권장 사항 단원을 참조하십시오.

IndexFaces 작업은 각 얼굴에 대해 다음의 정보를 유지합니다.

- 다차원 얼굴 특성 - IndexFaces는 얼굴 분석을 사용하여 얼굴 특성에 대한 다차원 정보를 추출하고, 이 정보를 얼굴 컬렉션에 저장합니다. 이 정보를 직접 액세스할 수는 없습니다. 그러나 Amazon Rekognition은 얼굴 컬렉션에서 얼굴 일치를 검색할 때 이 정보를 사용합니다.

- 메타데이터 – 각 얼굴의 메타데이터에는 경계 상자, 신뢰도 수준(경계 상자에 얼굴이 포함될 신뢰도 수준), Amazon Rekognition이 할당한 ID(얼굴 ID 및 이미지 ID), 요청 내 외부 이미지 ID(제공한 경우)가 포함됩니다. 이 정보는 IndexFaces API 호출에 대한 응답으로 반환됩니다. 관련 예제는 다음 응답 예의 face 요소를 참조하십시오.

  이 서비스는 다음 API 호출에 대한 응답으로 이 메타데이터를 반환합니다.

  - ListFaces
  - 얼굴 검색 작업 – SearchFaces 및 SearchFacesByImage에 대한 응답은 일치하는 얼굴의 일치 신뢰도와 함께 일치한 얼굴의 이 메타데이터를 반환합니다.

IndexFaces에서 인덱스를 생성하는 얼굴 수는 입력 모음과 연결된 얼굴 감지 모델의 버전에 따라 다릅니다. 자세한 내용은 모델 버전 관리 이해 단원을 참조하십시오.

인덱싱된 얼굴에 대한 정보는 FaceRecord 객체의 배열로 반환됩니다.

감지된 이미지와 인덱싱된 얼굴을 연결해야 합니다. 예를 들어 이미지와 이미지의 얼굴에서 클라이언트 측 인덱스를 유지해야 합니다. 얼굴을 이미지와 연결하려면, ExternalImageId 요청 파라미터에서 이미지 ID를 지정합니다. 이미지 ID는 만드는 파일 이름이나 다른 ID일 수 있습니다.

이 API는 얼굴 모음에서 유지하는 위의 정보 외에, 모음에 유지되지 않는 얼굴 세부 정보도 반환합니다. 다음 예제 응답에서 faceDetail 요소를 참조하십시오.

> **Note**
>
> DetectFaces는 동일한 정보를 반환하므로 같은 이미지에 대해 DetectFaces와 IndexFaces를 모두 호출할 필요는 없습니다.

얼굴 필터링

IndexFaces 작업을 통해 이미지에서 인덱싱된 얼굴을 필터링할 수 있습니다. IndexFaces를 통해 최대 얼굴 수를 인덱스로 지정하거나, 고화질로 감지된 얼굴만 인덱싱하도록 선택할 수 있습니다.

MaxFaces 입력 파라미터를 사용하여 IndexFaces로 인덱싱되는 최대 얼굴 수를 지정할 수 있습니다. 이는 이미지에서 가장 큰 얼굴을 인덱싱하고자 하며 작은 얼굴(예: 배경에 서 있는 사람들의 얼굴)은 인덱싱하지 않고자 할 때 유용합니다.

기본적으로 IndexFaces는 얼굴을 필터링하는 데 사용되는 화질 막대를 선택합니다. QualityFilter 입력 파라미터를 사용하여 화질 막대를 명시적으로 설정할 수 있습니다. 값은 다음과 같습니다.

- AUTO - Amazon Rekognition이 얼굴을 필터링하는 데 사용되는 화질 막대를 선택합니다(기본값).
- LOW - 가장 낮은 화질의 얼굴을 제외한 모든 얼굴이 인덱싱됩니다.
- MEDIUM
- HIGH - 가장 높은 화질의 얼굴만 인덱싱됩니다.
- NONE - 화질에 따라 얼굴이 필터링되지 않습니다.

IndexFaces는 다음과 같은 이유로 얼굴을 필터링합니다.

- 얼굴이 이미지 크기와 비교하여 너무 작습니다.
- 얼굴이 너무 흐릿합니다.
- 이미지가 너무 어둡습니다.
- 얼굴이 극단적인 포즈입니다.
- 얼굴에는 얼굴 검색에 적합한 세부 정보가 충분하지 않습니다.

> **Note**
>
> 화질 필터링을 사용하려면 얼굴 모델 버전 3 이상과 연결된 모음이 필요합니다. 컬렉션과 연결된 얼굴 모델 버전을 가져오려면 DescribeCollection을 직접 호출하세요.

IndexFaces로 인덱싱되지 않은 얼굴에 대한 정보는 UnindexedFace 객체의 배열로 반환됩니다. Reasons 배열에는 얼굴이 인덱싱되지 않은 이유 목록이 포함되어 있습니다. 예를 들어 MaxFaces로 지정된 얼굴 수가 이미 감지되었으므로 EXCEEDS_MAX_FACES 값은 인덱싱되지 않은 얼굴입니다.

자세한 내용은 컬렉션에서 얼굴 관리 단원을 참조하십시오.

모음에 얼굴을 추가하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.
   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.
   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. (하나 이상의 얼굴이 포함된) 이미지를 Amazon S3 버킷에 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.
3. 다음 예제를 사용하여 IndexFaces 작업을 호출합니다.

   Java

   이 예제는 모음에 추가된 얼굴의 얼굴 식별자를 표시합니다.

   collectionId의 값을, 얼굴을 추가하려는 모음의 이름으로 변경합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다. .withMaxFaces(1) 파라미터는 인덱싱되는 얼굴 수를 1로 제한합니다. 요구에 맞게 값을 제거하거나 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//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

이 AWS CLI 명령은 index-faces CLI 작업에 대한 JSON 출력을 표시합니다.

collection-id의 값을, 얼굴을 저장할 모음 이름으로 바꿉니다. Bucket 및 Name의 값을, 2단계에서 사용한 Amazon S3 버킷과 이미지 파일로 바꿉니다. max-faces 파라미터는 인덱싱되는 얼굴 수를 1로 제한합니다. 요구에 맞게 값을 제거하거나 변경합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
aws rekognition index-faces --image '{"S3Object":{"Bucket":"bucket-
name","Name":"file-name"}}' --collection-id "collection-id" \
                                --max-faces 1 --quality-filter "AUTO" --
detection-attributes "ALL" \
                                --external-image-id "example-image.jpg" --
profile profile-name
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
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

이 예제는 모음에 추가된 얼굴의 얼굴 식별자를 표시합니다.

collectionId의 값을, 얼굴을 추가하려는 모음의 이름으로 변경합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다. MaxFaces 입력 파라미터는 인덱싱되는 얼굴 수를 1로 제한합니다. 요구에 맞게 값을 제거하거나 변경합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

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

```
import boto3

def add_faces_to_collection(bucket, photo, collection_id):

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

    response = client.index_faces(CollectionId=collection_id,
                                  Image={'S3Object': {'Bucket': bucket, 'Name':
 photo}},
                                  ExternalImageId=photo,
                                  MaxFaces=1,
                                  QualityFilter="AUTO",
                                  DetectionAttributes=['ALL'])

    print('Results for ' + photo)
    print('Faces indexed:')
    for faceRecord in response['FaceRecords']:
        print('  Face ID: ' + faceRecord['Face']['FaceId'])
        print('  Location: {}'.format(faceRecord['Face']['BoundingBox']))

    print('Faces not indexed:')
    for unindexedFace in response['UnindexedFaces']:
        print(' Location: {}'.format(unindexedFace['FaceDetail']
['BoundingBox']))
        print(' Reasons:')
        for reason in unindexedFace['Reasons']:
            print('   ' + reason)
    return len(response['FaceRecords'])

def main():
    bucket = 'amzn-s3-demo-bucket'
    collection_id = 'collection-id'
    photo = 'photo-name'

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

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

.NET

이 예제는 모음에 추가된 얼굴의 얼굴 식별자를 표시합니다.

collectionId의 값을, 얼굴을 추가하려는 모음의 이름으로 변경합니다. bucket 및 photo의 값을 2단계에서 사용한 Amazon S3 버킷과 이미지의 이름으로 바꿉니다.

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

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

public class AddFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String bucket = "amzn-s3-demo-bucket";
        String photo = "input.jpg";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        Image image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo
            }
        };

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
        {
            Image = image,
            CollectionId = collectionId,
            ExternalImageId = photo,
            DetectionAttributes = new List<String>(){ "ALL" }
        };
```

```
        IndexFacesResponse indexFacesResponse =
 rekognitionClient.IndexFaces(indexFacesRequest);

        Console.WriteLine(photo + " added");
        foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
            Console.WriteLine("Face detected: Faceid is " +
               faceRecord.Face.FaceId);
    }
}
```

IndexFaces 작업 요청

IndexFaces에 대한 입력은 인덱스를 생성할 이미지와, 얼굴을 추가할 모음입니다.

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "ExternalImageId": "input.jpg",
    "DetectionAttributes": [
        "DEFAULT"
    ],
    "MaxFaces": 1,
    "QualityFilter": "AUTO"
}
```

IndexFaces 작업 응답

IndexFaces는 이미지에서 감지된 얼굴에 대한 정보를 반환합니다. 예를 들어 다음 JSON 응답에는 입력 이미지에서 감지된 얼굴의 기본 감지 속성이 포함됩니다. 이 예제에서는 MaxFaces 입력 파라미터의 값이 초과되어 인덱싱되지 않은 얼굴도 보여줍니다. Reasons 배열에는 EXCEEDS_MAX_FACES가 포함되어 있습니다. 화질로 인해 얼굴이 인덱싱되지 않은 경우 Reasons에는 LOW_SHARPNESS 또는 LOW_BRIGHTNESS 등의 값이 포함되어 있습니다. 자세한 내용은 UnindexedFace를 참조하십시오.

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

모든 얼굴 정보를 가져오려면 DetectionAttributes 요청 파라미터에서 'ALL'을 지정합니다. 예를 들어 다음 예제 응답에서 faceDetail 요소의 추가 정보에 유의해야 하며 이것은 서버에서 지속되지 않습니다.

- 25개의 얼굴 표식(앞의 예의 단 5개와 비교)
- 10개의 얼굴 속성(안경, 수염, 오클루전, 시선 방향 등)
- 감정(emotion 요소 참조)

face 요소는 서버에서 유지되는 메타데이터를 제공합니다.

FaceModelVersion은 모음과 연결된 얼굴 모델의 버전입니다. 자세한 내용은 모델 버전 관리 이해 단원을 참조하십시오.

OrientationCorrection은 이미지의 예상 방향입니다. 버전 3보다 높은 얼굴 감지 모델을 사용하는 경우 방향 보정 정보가 반환되지 않습니다. 자세한 내용은 이미지 방향 및 경계 상자 좌표 가져오기 단원을 참조하십시오.

다음 샘플 응답은 ["ALL"] 을 지정할 때 반환된 JSON을 보여줍니다.


```
{
    "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": []
}
```

컬렉션의 얼굴 및 연결된 사용자 나열

ListFaces 작업을 사용하여 컬렉션의 얼굴과 그에 연결된 사용자를 나열할 수 있습니다.

자세한 내용은 컬렉션에서 얼굴 관리 단원을 참조하십시오.

모음에 있는 얼굴을 나열하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.
   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.
   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제를 사용하여 ListFaces 작업을 호출합니다.

   Java

   이 예제는 모음의 얼굴 목록을 표시합니다.

   collectionId의 값을 원하는 모음으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
// 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

이 AWS CLI 명령은 `list-faces` CLI 작업에 대한 JSON 출력을 표시합니다. `collection-id`의 값을, 목록을 조회할 모음의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
aws rekognition list-faces --collection-id "collection-id"  --profile profile-
name
```

Python

이 예제는 모음의 얼굴 목록을 표시합니다.

Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 모음의 얼굴 목록을 표시합니다.

`collectionId`의 값을 원하는 모음으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

ListFaces에 대한 입력은 얼굴 목록을 조회하려는 모음의 ID입니다. MaxResults는 반환할 최대 얼굴 수입니다. 또한 ListFaces는 결과를 필터링할 때 사용할 얼굴 ID 목록과 지정된 사용자와 연결된 얼굴만 나열하기 위해 제공된 사용자 ID를 가져옵니다.

```
{
    "CollectionId": "MyCollection",
    "MaxResults": 1
}
```

응답의 얼굴이, MaxResults가 요청한 것보다 많으면 후속 ListFaces 호출에서 다음 결과 집합을 가져오는 데 사용할 수 있는 토큰이 반환됩니다. 예제:

```
{
    "CollectionId": "MyCollection",
    "NextToken": "sm+5ythT3aeEVIR4WA....",
    "MaxResults": 1
}
```

ListFaces 작업 응답

ListFaces의 응답은 지정한 모음에 저장된 얼굴 메타데이터에 대한 정보입니다.

- FaceModelVersion – 컬렉션과 연결된 얼굴 모델의 버전입니다. 자세한 내용은 모델 버전 관리 이해 단원을 참조하십시오.

- Faces – 컬렉션에 있는 얼굴에 대한 정보입니다. 이에는 BoundingBox, 신뢰도, 이미지 식별자 및 얼굴 ID에 관한 정보가 포함됩니다. 자세한 내용은 Face를 참조하십시오.
- NextToken – 다음 결과 집합을 가져오는 데 사용되는 토큰입니다.


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

컬렉션에서 얼굴 삭제

DeleteFaces 작업을 사용하여 컬렉션에서 얼굴을 삭제할 수 있습니다. 자세한 내용은 컬렉션에서 얼굴 관리 단원을 참조하십시오.

컬렉션에서 얼굴 삭제

1. 아직 설정하지 않았다면 다음과 같이 하세요.
   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.
   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제를 사용하여 DeleteFaces 작업을 호출합니다.

   Java

   이 예제는 모음에서 한 개의 얼굴을 삭제합니다.

   collectionId의 값을, 삭제하려는 얼굴을 포함하는 모음으로 변경합니다. faces의 값을, 삭제하려는 얼굴 ID로 변경합니다. 여러 얼굴을 삭제하려면 faces 배열에 얼굴 ID를 추가합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
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

이 AWS CLI 명령은 delete-faces CLI 작업에 대한 JSON 출력을 표시합니다. collection-id의 값을, 삭제하려는 얼굴을 포함하는 모음의 이름으로 바꿉니다. face-ids의 값을, 삭제하려는 얼굴 ID 배열로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
aws rekognition delete-faces --collection-id "collection-id" --face-ids "faceid"
 --profile profile-name
```

Python

이 예제는 모음에서 한 개의 얼굴을 삭제합니다.

collectionId의 값을, 삭제하려는 얼굴을 포함하는 모음으로 변경합니다. faces의 값을, 삭제하려는 얼굴 ID로 변경합니다. 여러 얼굴을 삭제하려면 faces 배열에 얼굴 ID를 추가합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 모음에서 한 개의 얼굴을 삭제합니다.

`collectionId`의 값을, 삭제하려는 얼굴을 포함하는 모음으로 변경합니다. `faces`의 값을, 삭제하려는 얼굴 ID로 변경합니다. 여러 얼굴을 삭제하려면 `faces` 목록에 얼굴 ID를 추가합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

DeleteFaces에 대한 입력은 얼굴을 포함하는 모음의 ID와, 삭제할 얼굴의 얼굴 ID 배열입니다.

```
{
    "CollectionId": "MyCollection",
    "FaceIds": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

DeleteFaces 작업 응답

DeleteFaces 응답에는 삭제된 얼굴의 얼굴 ID 배열이 들어 있습니다.

```
{
    "DeletedFaces": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

입력에 제공된 얼굴 ID가 현재 사용자와 연결되어 있는 경우 타당한 이유가 있는 UnsuccessfulFaceDeletions의 일부로 반환됩니다.

```
{
   "DeletedFaces": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ],
     "UnsuccessfulFaceDeletions" : [
        {
           "FaceId" : "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
           "UserId" : "demoUser1",
           "Reason" : ["ASSOCIATED_TO_AN_EXISTING_USER"]
        }
     ]
}
```

사용자 생성

CreateUser 작업을 사용하여 귀하가 제공하는 고유한 사용자 ID로 컬렉션에 새 사용자를 생성할 수 있습니다. 그런 다음 새로 만든 사용자와 얼굴 여러 개를 연결할 수 있습니다.

사용자를 생성하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한이 있는 IAM 사용자 계정을 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 CreateUser 작업을 호출합니다.

   Java

   이 Java 코드 예제는 사용자를 생성합니다.

```
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

이 AWS CLI 명령은 `create-user` CLI 작업을 사용하여 사용자를 생성합니다.

```
aws rekognition create-user --user-id user-id --collection-id collection-name --
region region-name
--client-request-token request-token
```

Python

이 Python 코드 예제는 사용자를 생성합니다.

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

import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def create_user(collection_id, user_id):
    """
    Creates a new User within a collection specified by CollectionId.
    Takes UserId as a parameter, which is a user provided ID which
    should be unique within the collection.

    :param collection_id: The ID of the collection where the indexed faces will
 be stored at.
    :param user_id: ID for the UserID to be created. This ID needs to be unique
 within the collection.

    :return: The indexFaces response
    """
    try:
        logger.info(f'Creating user: {collection_id}, {user_id}')
        client.create_user(
            CollectionId=collection_id,
            UserId=user_id
        )
    except ClientError:
        logger.exception(f'Failed to create user with given user id:
 {user_id}')
        raise

def main():
    collection_id = "collection-id"
    user_id = "user-id"
    create_user(collection_id, user_id)

if __name__ == "__main__":
```

```
    main()
```

Go

이 Go 코드 예제에서는 AWS Go SDK V2를 사용하고 사용자를 생성합니다.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rekognition"
)

func main() {
 // Load the AWS SDK configuration
 cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-
west-2"))
 if err != nil {
  log.Fatalf("Failed to load configuration: %v", err)
 }

 // Create a Rekognition client
 client := rekognition.NewFromConfig(cfg)

 // Set up the input parameters
 input := &rekognition.CreateUserInput{
  CollectionId: aws.String("my-new-collection"), // Replace with your collection
 ID
  UserId:       aws.String("user12345678910"),   // Replace with desired user ID
 }

 // Call the CreateUser operation
 result, err := client.CreateUser(context.TODO(), input)
 if err != nil {
  log.Fatalf("Failed to create user: %v", err)
 }
```

```
 // Print out the results
 fmt.Printf("User created successfully:\n")

}
```

사용자 삭제

DeleteUser 작업을 사용하여 제공된 UserID를 기반으로 컬렉션에서 사용자를 삭제할 수 있습니다. UserID와 연결된 모든 얼굴은 해당 UserID가 삭제되기 전에 UserID와의 연결이 해제된다는 점에 유의하십시오.

사용자를 삭제하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한이 있는 IAM 사용자 계정을 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 DeleteUser 작업을 호출합니다.

   Java

   이 Java 코드 예제는 사용자를 삭제합니다.

```
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

이 AWS CLI 명령은 `create-user` CLI 작업을 사용하여 사용자를 삭제합니다.

```
aws rekognition delete-user --collection-id MyCollection
--user-id user-id --collection-id collection-name --region region-name
```

Python

이 Python 코드 예제는 사용자를 삭제합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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()
```

얼굴을 사용자에 연결

AssociateFaces 작업을 사용하여 여러 개별 얼굴을 단일 사용자와 연결할 수 있습니다. 얼굴을 사용자와 연결하려면 먼저 컬렉션과 사용자를 생성해야 합니다. 단, 얼굴 벡터는 사용자 벡터가 있는 컬렉션과 동일한 컬렉션에 있어야 합니다.

얼굴을 연결하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 AssociateFaces 작업을 호출합니다.

Java

이 Java 코드 예제는 얼굴을 사용자와 연결합니다.

```
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

이 AWS CLI 명령은 associate-faces CLI 작업을 사용하여 얼굴을 사용자와 연결합니다.

```
aws rekognition associate-faces --user-id user-id --face-ids face-id-1 face-id-2
--collection-id collection-name
--region region-name
```

Python

이 Python 코드 예제는 얼굴을 사용자와 연결합니다.

```
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 작업 응답

AssociateFaces에 대한 응답에는 연결 해제 요청의 상태인 UserStatus 및 연결할 FaceIds 목록이 포함됩니다. UnsuccessfulFaceAssociations 목록도 반환됩니다. AssociateFaces에 요청을 제출한 후 작업을 완료하는 데 1분 정도 걸릴 수 있습니다.

이러한 이유로 다음과 같은 값을 가질 수 있는 UserStatus가 반환됩니다.

- CREATED - 'User'가 성공적으로 생성되었으며 현재 연결된 얼굴이 없음을 나타냅니다. 'AssociateFaces' 직접 호출이 성공적으로 이루어지기 전에는 'User'가 이 상태에 있게 됩니다.

- UPDATING - 'User'가 새로 연결되었거나 연결 해제된 얼굴을 반영하도록 업데이트되고 있으며 몇 초 후에 ACTIVE 상태가 될 것을 나타냅니다. 이 상태에서는 검색 결과에 'User'가 포함될 수 있으며 고객은 반환된 결과에서 해당 사용자를 무시하도록 선택할 수 있습니다.
- ACTIVE - 'User'가 연결되거나 연결 해제된 모든 얼굴을 반영하도록 업데이트되었으며 검색 가능한 상태임을 나타냅니다.


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


사용자에게서 얼굴 연결 해제

DisassociateFaces 작업을 사용하여 사용자 ID와 얼굴 ID 간의 연결을 제거할 수 있습니다.

얼굴을 연결 해제하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 DisassociateFaces 작업을 호출합니다.

Java

이 Java 예제는 DisassociateFaces 작업을 사용해 FaceID와 UserID 간의 연결을 제거합니다.

```
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

이 AWS CLI 명령은 FaceID와 UserID 간의 연결을 `DisassociateFaces` 작업과 함께 제거합니다.

```
aws rekognition disassociate-faces --face-ids list-of-face-ids
--user-id user-id --collection-id collection-name --region region-name
```

Python

다음 예제는 `DisassociateFaces` 작업을 사용해 FaceID와 UserID 간의 연결을 제거합니다.

```
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 작업 응답

DisassociateFaces에 대한 응답에는 연결 해제 요청의 상태인 UserStatus 및 연결 해제 할 FaceIds 목록이 포함됩니다. UnsuccessfulFaceDisassociations 목록도 반환됩니다. DisassociateFaces에 요청을 제출한 후 작업을 완료하는 데 1분 정도 걸릴 수 있습니다. 이러한 이유로 다음과 같은 값을 가질 수 있는 UserStatus가 반환됩니다.

- CREATED - 'User'가 성공적으로 생성되었으며 현재 연결된 얼굴이 없음을 나타냅니다. 'AssociateFaces' 직접 호출이 성공적으로 이루어지기 전에는 'User'가 이 상태에 있게 됩니다.
- UPDATING - 'User'가 새로 연결되었거나 연결 해제된 얼굴을 반영하도록 업데이트되고 있으며 몇 초 후에 ACTIVE 상태가 될 것을 나타냅니다. 이 상태에서는 검색 결과에 'User'가 포함될 수 있으며 고객은 반환된 결과에서 해당 사용자를 무시하도록 선택할 수 있습니다.
- ACTIVE - 'User'가 연결되거나 연결 해제된 모든 얼굴을 반영하도록 업데이트되었으며 검색 가능한 상태임을 나타냅니다.


```
{
    "UserStatus": "UPDATING",
    "DisassociatedFaces": [
        {
            "FaceId": "c92265d4-5f9c-43af-a58e-12be0ce02bc3"
        }
    ],
    "UnsuccessfulFaceDisassociations": [
        {
            "Reasons": [
                "ASSOCIATED_TO_A_DIFFERENT_IDENTITY"
            ],
            "FaceId": "f5817d37-94f6-4335-bfee-6cf79a3d806e",
            "UserId": "demoUser1"
        }
    ]
}
```


컬렉션에서 사용자 나열

ListUsers 작업을 사용하여 UserIds 및 UserStatus를 나열할 수 있습니다. UserID와 연결된 FaceID를 보려면 ListFaces 작업을 사용하십시오.

사용자를 나열하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 ListUsers 작업을 호출합니다.

Java

이 Java 예제는 ListUsers 작업을 사용해 컬렉션에 있는 사용자를 나열합니다.

```
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

이 AWS CLI 명령은 `ListUsers` 작업이 있는 컬렉션의 사용자를 나열합니다.

```
aws rekognition list-users --collection-id collection-id --max-results number-
of-max-results
```

Python

다음 예제에서는 `ListUsers` 작업으로 컬렉션에 있는 사용자를 나열합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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 작업 응답

ListUsers에 보낸 요청에 대한 응답에는 컬렉션의 Users 목록과 User의 UsedId 및 UserStatus가 포함됩니다.

```
{
    "NextToken": "B1asJT3bAb/ttuGgPFV8BZoBZyGQzlUHXbuTNLh48a6enU7kXKw43hpOwizW7LOk/
Gk7Em09lznoq6+FcDCcSq2olrn7A98BLkt5keu+ZRVRUTyrXtT6J7Hmp
+ieQ2an6Zu0qzPfcdPeaJ9eAxG2d0WNrzJgi5hvmjoiSTTfKX3MQz1sduWQkvAAs4hZfhZoKFahFlqWofshCXa/
FHAAY3PL1PjxXbkNeSSMq8V7i1MlKCdrPVykCv9MokpPt7jtNvKPEZGUhxgBTFMxNWLEcFnzAiCWDg91dFy/
LalshPjXA9UVc5Gx9vIJNQ/
eO3cQRghAkCT3FOAiXsLAnA015ODTomZpWWVpqB21wKpI3LYmfAVFrDPGzpbTVlRmLsJm41bkmnBBBw9+DHz1Jn7zW
+qc5Fs3yaHu0f51Xg==",
    "Users": [
        {
            "UserId": "demoUser4",
            "UserStatus": "CREATED"
        },
        {
            "UserId": "demoUser2",
            "UserStatus": "CREATED"
        }
    ]
}
```

얼굴 ID로 얼굴 검색

SearchFaces 작업을 사용하여 제공된 이미지에서 가장 큰 얼굴과 일치하는 컬렉션에서 사용자를 검색할 수 있습니다.

얼굴이 감지되고 컬렉션에 추가되면 IndexFaces 작업 응답에서 얼굴 ID가 반환됩니다. 자세한 내용은 컬렉션에서 얼굴 관리 단원을 참조하십시오.

얼굴 ID(SDK)를 사용하여 모음에서 얼굴을 검색하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.
   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.
   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제를 사용하여 SearchFaces 작업을 호출합니다.

   Java

   이 예제는 ID로 식별한 얼굴과 일치하는 얼굴에 대한 정보를 표시합니다.

   collectionID의 값을, 원하는 얼굴이 있는 모음으로 변경합니다. faceId의 값을, 찾으려는 얼굴의 식별자로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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();
      }
    }

}
```

예제 코드를 실행합니다. 일치하는 얼굴에 대한 정보가 표시됩니다.

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
// 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

이 AWS CLI 명령은 `search-faces` CLI 작업에 대한 JSON 출력을 표시합니다. 검색하려는 얼굴 식별자로 `face-id` 값을 바꾸고, 검색하려는 모음으로 `collection-id`를 바꿉니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
aws rekognition search-faces --face-id face-id --collection-id "collection-id"
 --profile profile-name
```

Python

이 예제는 ID로 식별한 얼굴과 일치하는 얼굴에 대한 정보를 표시합니다.

`collectionID`의 값을, 원하는 얼굴이 있는 모음으로 변경합니다. `faceId`의 값을, 찾으려는 얼굴의 식별자로 변경합니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 ID로 식별한 얼굴과 일치하는 얼굴에 대한 정보를 표시합니다.

`collectionID`의 값을, 원하는 얼굴이 있는 모음으로 변경합니다. `faceId`의 값을, 찾으려는 얼굴의 식별자로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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);
    }
}
```

예제 코드를 실행합니다. 일치하는 얼굴에 대한 정보가 표시됩니다.

SearchFaces 작업 요청

얼굴 ID(얼굴 모음에 저장되는 각 얼굴은 얼굴 ID를 가짐)를 지정할 경우, SearchFaces는 지정된 얼굴 모음에서 유사한 얼굴을 검색합니다. 응답에 검색하는 얼굴이 포함되지 않았습니다. 유사한 얼굴만이 포함됩니다. SearchFaces는 기본적으로 알고리즘이 80% 이상의 유사성을 감지하는 얼굴을 반환합니다. 유사성은 얼굴이 입력 얼굴과 얼마나 일치하는지를 나타냅니다. 필요하다면 FaceMatchThreshold를 사용하여 다른 값을 지정할 수 있습니다.

```
{
    "CollectionId": "MyCollection",
    "FaceId": "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFaces 작업 응답

작업은 발견된 얼굴 일치의 배열과 입력으로 제공한 얼굴 ID를 반환합니다.

```
{
    "SearchedFaceId": "7ecf8c19-5274-5917-9c91-1db9ae0449e2",
    "FaceMatches": [ list of face matches found ]
}
```

발견된 얼굴 일치마다 응답에는 유사성과 얼굴 메타데이터가 포함됩니다. 다음 예제 응답을 참조하십시오.

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


이미지를 사용하여 얼굴 검색

SearchFacesByImage 작업을 사용하여 제공된 이미지에서 가장 큰 얼굴과 일치하는 컬렉션에서 얼굴을 검색할 수 있습니다.

자세한 내용은 컬렉션 내의 얼굴 및 사용자 검색 단원을 참조하십시오.

이미지를 사용하여 모음에서 얼굴을 검색하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 한 개 이상의 얼굴이 포함된 이미지를 S3 버킷에 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 예제를 사용하여 SearchFacesByImage 작업을 호출합니다.

   Java

   이 예제는 이미지에서 가장 큰 얼굴에 대한 정보를 표시합니다. 이 코드 예제는 FaceMatchThreshold 및 MaxFaces 파라미터를 지정하여 응답에서 반환되는 결과를 제한합니다.

   다음 예제에서, collectionId 값을, 검색하려는 모음으로 바꾸고, bucket 값을 입력 이미지를 포함하는 버킷으로 바꾸고, photo 값을 입력 이미지로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
// 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

이 AWS CLI 명령은 `search-faces-by-image` CLI 작업에 대한 JSON 출력을 표시합니다. `Bucket` 값을 2단계에서 사용한 S3 버킷으로 바꿉니다. `Name` 값을 2단계에서 이미지 파일 이름으로 바꿉니다. `collection-id` 값을 검색할 모음으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
aws rekognition search-faces-by-image --image '{"S3Object":{"Bucket":"bucket-
name","Name":"image-name"}}' \
--collection-id "collection-id" --profile profile-name
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
aws rekognition search-faces-by-image --image "{\"S3Object\":{\"Bucket\":
\"bucket-name\",\"Name\":\"image-name\"}}" \
--collection-id "collection-id" --profile profile-name
```

Python

이 예제는 이미지에서 가장 큰 얼굴에 대한 정보를 표시합니다. 이 코드 예제는 FaceMatchThreshold 및 MaxFaces 파라미터를 지정하여 응답에서 반환되는 결과를 제한합니다.

다음 예제에서, collectionId 값을 검색하려는 컬렉션으로 바꾸고, bucket 값과 photo 값을 2단계에서 사용한 Amazon S3 버킷 및 이미지 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

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

import boto3

if __name__ == "__main__":

    bucket='amzn-s3-demo-bucket'
    collectionId='MyCollection'
    fileName='input.jpg'
    threshold = 70
    maxFaces=2

    client=boto3.client('rekognition')


    response=client.search_faces_by_image(CollectionId=collectionId,
                                Image={'S3Object':
{'Bucket':bucket,'Name':fileName}},
                                FaceMatchThreshold=threshold,
                                MaxFaces=maxFaces)


    faceMatches=response['FaceMatches']
    print ('Matching faces')
    for match in faceMatches:
            print ('FaceId:' + match['Face']['FaceId'])
            print ('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
            print
```

.NET

이 예제는 이미지에서 가장 큰 얼굴에 대한 정보를 표시합니다. 이 코드 예제는 FaceMatchThreshold 및 MaxFaces 파라미터를 지정하여 응답에서 반환되는 결과를 제한합니다.

다음 예제에서, collectionId 값을 검색하려는 컬렉션으로 바꾸고, bucket 값과 photo 값을 2단계에서 사용한 Amazon S3 버킷 및 이미지 이름으로 바꿉니다.

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

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

public class SearchFacesMatchingImage
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String bucket = "amzn-s3-demo-bucket";
        String photo = "input.jpg";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        Image image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo
            }
        };

        SearchFacesByImageRequest searchFacesByImageRequest = new
 SearchFacesByImageRequest()
        {
            CollectionId = collectionId,
```

```
            Image = image,
            FaceMatchThreshold = 70F,
            MaxFaces = 2
        };

        SearchFacesByImageResponse searchFacesByImageResponse =
 rekognitionClient.SearchFacesByImage(searchFacesByImageRequest);

        Console.WriteLine("Faces matching largest face in image from " + photo);
        foreach (FaceMatch face in searchFacesByImageResponse.FaceMatches)
            Console.WriteLine("FaceId: " + face.Face.FaceId + ", Similarity: " +
 face.Similarity);
    }
}
```

SearchFacesByImage 작업 요청

SearchFacesImageByImage에 대한 입력 파라미터는 검색할 모음과 소스 이미지 위치입니다. 이 예제에서 소스 이미지는 Amazon S3 버킷(S3Object)에 저장되어 있습니다. 또한 반환할 최대 얼굴 수(Maxfaces)와, 반환할 얼굴에 대해 일치해야 할 최소 신뢰도(FaceMatchThreshold)를 지정합니다.

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFacesByImage 작업 응답

입력 이미지(.jpeg 또는 .png)가 주어진 경우, 작업은 먼저 입력 이미지에서 얼굴을 감지한 다음 지정된 얼굴 모음에서 유사한 얼굴을 감지합니다.

> **Note**
>
> 입력 이미지에서 여러 얼굴이 감지되는 경우, 서비스는 감지된 가장 큰 얼굴을 얼굴 모음 감지에 사용합니다.

이 작업은 찾아낸 일치 얼굴의 배열과, 입력 얼굴에 대한 정보를 반환합니다. 경계 상자 등과 같은 정보와 신뢰도 값을 포함합니다. 신뢰도는 경계 상자의 얼굴 포함 기준이 되는 신뢰도 수준입니다.

SearchFacesByImage는 기본적으로 알고리즘이 80% 이상의 유사성을 감지하는 얼굴을 반환합니다. 유사성은 얼굴이 입력 얼굴과 얼마나 일치하는지를 나타냅니다. 필요하다면 FaceMatchThreshold를 사용하여 다른 값을 지정할 수 있습니다. 발견된 얼굴 일치마다 응답에는 유사성과 얼굴 메타데이터가 포함됩니다. 다음 예제 응답을 참조하십시오.


```
{
    "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
}
```

사용자 검색(얼굴 ID 또는 사용자 ID)

SearchUsers 작업을 사용하여 제공된 얼굴 ID 혹은 사용자 ID와 일치하는 컬렉션을 지정하여 사용자를 검색할 수 있습니다. 이 작업은 반환된 UserIds 결과를 요청한 UserMatchThreshold보다 높은 유사성 점수 순대로 나열합니다. 사용자 ID는 CreateUsers 작업에서 생성됩니다. 자세한 내용은 컬렉션의 사용자 관리 단원을 참조하십시오.

사용자 검색(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 SearchUsers 작업을 호출합니다.

   Java

   이 Java 예제는 SearchUsers 작업을 사용하여 컬렉션에서 사용자를 검색합니다.

```
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

이 AWS CLI 명령은 SearchUsers 작업을 사용하여 컬렉션의 사용자를 검색합니다.

```
aws rekognition search-users --face-id face-id --collection-id collection-id --
region region-name
```

Python

다음 예제에서는 SearchUsers 작업으로 컬렉션에 있는 사용자를 검색합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

FaceID 또는 UserID가 주어지면 SearchUsers는 지정된 CollectionID에서 일치하는 사용자를 검색합니다. 기본적으로 SearchUsers는 유사성 점수가 80%를 초과하는 UserID를 반환합니다. 유사성은 UserID가 제공된 FaceID 또는 UserID와 얼마나 가깝게 일치하는지를 나타냅니다. 여러 UserID가 반환되는 경우 유사성 점수가 높은 것부터 가장 낮은 순으로 나열됩니다. 필요하다면 UserMatchThreshold를 사용하여 다른 값을 지정할 수 있습니다. 자세한 내용은 컬렉션의 사용자 관리 단원을 참조하십시오.

다음은 `UserId`를 사용하는 SearchUsers 요청의 예입니다.

```
{
   "CollectionId": "MyCollection",
   "UserId": "demoUser1",
   "MaxUsers": 2,
   "UserMatchThreshold": 99
}
```

다음은 FaceId를 사용하는 SearchUsers 요청의 예입니다.

```
{
    "CollectionId": "MyCollection",
    "FaceId": "bff43c40-cfa7-4b94-bed8-8a08b2205107",
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsers 작업 응답

FaceId로 검색하는 경우 SearchUsers에 대한 응답에는 SearchedFace에 대한 FaceId와 UserMatches 목록 및 각 User의 UserId와UserStatus가 포함됩니다.

```
{
    "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"
}
```

UserId로 검색하는 경우 SearchUsers에 대한 응답에는 다른 응답 요소와 함께 SearchedUser에 대한 UserId가 포함됩니다.

```
{
    "SearchedUser": {
        "UserId": "demoUser1"
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6"
}
```

사용자 검색(이미지)

SearchUsersByImage는 지정된 CollectionID를 검색하여 제공된 이미지에서 감지된 가장 큰 얼굴과 일치하는 컬렉션의 사용자를 찾습니다. 기본적으로 SearchUsersByImage는 유사성 점수가 80%를 초과하는 UserID를 반환합니다. 유사성은 UserID가 제공된 이미지에서 감지된 가장 큰 얼굴과 얼마나 가깝게 일치하는지를 나타냅니다. 여러 UserID가 반환되는 경우 유사성 점수가 높은 것부터 가장 낮은 순으로 나열됩니다. 필요하다면 UserMatchThreshold를 사용하여 다른 값을 지정할 수 있습니다. 자세한 내용은 the section called “컬렉션의 사용자 관리” 단원을 참조하십시오.

이미지로 사용자 검색하기(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 SearchUsersByImage 작업을 호출합니다.

   Java

   이 Java 예제는 SearchUsersByImage 작업을 사용하여 입력 이미지를 기반으로 컬렉션에 있는 사용자를 검색합니다.

```
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

이 AWS CLI 명령은 SearchUsersByImage 작업을 사용하여 입력 이미지를 기반으로 컬렉션의 사용자를 검색합니다.

```
aws rekognition search-users-by-image --image '{"S3Object":
{"Bucket":"s3BucketName","Name":"file-name"}}' --collection-id MyCollectionId --
region region-name
```

Python

다음 예제에서는 SearchUsersByImage 작업을 사용하여 입력 이미지를 기반으로 컬렉션에 있는 사용자를 검색합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

SearchUsersByImage에 대한 요청은 검색할 컬렉션과 소스 이미지 위치를 포함합니다. 이 예제에서 소스 이미지는 Amazon S3 버킷(S3Object)에 저장되어 있습니다. 또한 반환할 최대 사용자 수(MaxUsers)와, 반환할 사용자에 대해 일치해야 하는 최소 신뢰도(UserMatchThreshold)를 지정합니다.

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsersByImage 작업 응답

SearchUsersByImage에 대한 응답에는 SearchedFace에 대한 FaceDetail 개체와 UserMatches 목록 및 각각의 UserId, Similarity, 그리고 UserStatus가 포함됩니다. 입력 이미지에 얼굴이 두 개 이상 포함된 경우 UnsearchedFaces 목록도 반환됩니다.

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

저장된 비디오에서 얼굴 검색

저장된 비디오나 스트리밍 비디오에서 감지한 얼굴과 일치하는 얼굴을 얼굴 모음에서 찾아낼 수 있습니다. 이 단원에서는 저장된 비디오에서 얼굴을 검색하는 과정에 대해 다룹니다. 스트리밍 비디오에서 얼굴을 검색하는 방법에 대한 정보는 스트리밍 비디오 이벤트 작업를 참조하십시오.

먼저 IndexFaces를 사용하여 검색할 얼굴을 컬렉션으로 인덱싱해야 합니다. 자세한 내용은 컬렉션에 얼굴 추가 단원을 참조하십시오.

Amazon Rekognition Video 얼굴 검색은 Amazon S3 버킷에 저장된 비디오를 분석하는 다른 Amazon Rekognition Video 작업과 동일한 비동기 워크플로를 따릅니다. 저장된 비디오에서 얼굴 검색을 시작하려면 StartFaceSearch를 직접 호출하고 검색하려는 컬렉션의 ID를 제공하십시오. Amazon Rekognition Video는 동영상 분석 작업의 완료 상태를 Amazon Simple Notification Service(SNS) 주제에 게시합니다. 비디오 분석이 성공적으로 완료되면 GetFaceSearch를 직접 호출하여 검색 결과를 가져옵니다. 비디오 분석 시작 및 결과 가져오기에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오.

다음 절차는 비디오에서 감지된 사람의 얼굴과 일치하는 얼굴 모음을 감지하는 방법을 보여 줍니다. 다음 절차에서는 비디오에서 일치되는 사람의 추적 데이터를 가져오는 방법을 보여 줍니다. 이 절차는 동영상 분석 요청의 완료 상태를 가져오기 위해 Amazon Simple Queue Service(Amazon SQS) 대기열을 사용하는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)의 코드를 확장합니다.

비디오에서 일치하는 얼굴을 검색하려면(SDK)

1. 모음을 만듭니다.
2. 얼굴을 모음으로 인덱싱합니다.
3. Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)을 수행합니다.
4. 3단계에서 만든 클래스 `VideoDetect`에 다음 코드를 추가합니다.

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);

    }
```

main 함수에서 다음 줄을 바꿉니다.

```
        StartLabelDetection(bucket, video);

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

다음으로 바꿉니다.

```
        String collection="collection";
        StartFaceSearchCollection(bucket, video, collection);

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

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
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()
```

`main` 함수에서 다음 줄을 바꿉니다.

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

다음으로 바꿉니다.

```
    collection='tests'
    analyzer.StartFaceSearchCollection(collection)

    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetFaceSearchCollectionResults()
```

Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 이외에 비디오 예제를 이미 실행한 경우, 바꿀 코드가 다를 수 있습니다.

5. `collection`의 값을, 1단계에서 만든 모음의 이름으로 변경합니다.
6. 코드를 실행합니다. 입력 모음의 얼굴과 일치하는 비디오의 사람 목록이 표시됩니다. 일치하는 각 사람의 추적 데이터도 표시됩니다.

GetFaceSearch 작업 응답

다음은 GetFaceSearch의 JSON 응답 예제입니다.

응답에는 입력 모음의 얼굴과 얼굴이 일치하는 동영상에서 감지된 일련의 사람 Persons이 포함됩니다. 배열 요소 PersonMatch는 비디오에서 사람이 일치할 때마다 존재합니다. 각PersonMatch에는 입력 컬렉션에서 가져오는 일련의 얼굴 일치, FaceMatch, 일치하는 사람에 관한 정보, PersonDetail 및 이 사람이 비디오에서 일치된 시간이 포함되어 있습니다.


```
{
    "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
    }
}
```

스트리밍 비디오에서 컬렉션의 얼굴 검색

Amazon Rekognition Video를 사용하면 스트리밍 비디오의 컬렉션에서 얼굴을 감지하고 인식할 수 있습니다. Amazon Rekognition Video를 사용하면 스트림 프로세서(CreateStreamProcessor)를 생성하여 스트리밍 비디오 분석을 시작하고 관리할 수 있습니다.

비디오 스트림에서 알려진 얼굴을 감지(얼굴 검색)하기 위해 Amazon Rekognition Video는 Amazon Kinesis Video Streams를 사용하여 비디오 스트림을 수신하고 처리합니다. 분석 결과는 Amazon Rekognition Video에서 Kinesis 데이터 스트림으로 출력된 후 클라이언트 애플리케이션이 읽게 됩니다.

스트리밍 비디오에 Amazon Rekognition Video를 사용하려면 애플리케이션에 다음을 구현해야 합니다.

- Amazon Rekognition Video로 스트리밍 비디오를 전송하기 위한 Kinesis 비디오 스트림. 자세한 내용은 Amazon Kinesis Video Streams 개발자 안내서를 참조하세요.
- 스트리밍 비디오 분석을 관리할 Amazon Rekognition Video 스트림 프로세서. 자세한 내용은 Amazon Rekognition Video 스트림 프로세서 작업 개요 단원을 참조하십시오.
- Amazon Rekognition Video가 Kinesis 데이터 스트림으로 전송하는 분석 결과를 읽을 Kinesis 데이터 스트림 소비자. 자세한 내용은 Amazon Kinesis Data Streams 소비자를 참조하세요.

이 섹션에는 Kinesis 비디오 스트림 및 기타 필요한 리소스를 생성하고, Amazon Rekognition Video로 비디오를 스트리밍하고, 분석 결과를 수신하는 애플리케이션을 개발하는 방법에 대한 정보가 포함되어 있습니다.

주제

- Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정
- 스트리밍 비디오에서 얼굴 검색
- GStreamer 플러그인을 사용한 스트리밍
- 스트리밍 비디오 문제 해결

Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정

다음 절차는 스트리밍 비디오에서 얼굴을 인식하는 데 사용되는 Kinesis 비디오 스트림 및 기타 리소스를 프로비저닝하기 위해 수행할 단계를 설명합니다.

사전 조건

이 절차를 실행하려면가 AWS SDK for Java 설치되어 있어야 합니다. 자세한 내용은 Amazon Rekognition 시작 단원을 참조하십시오. AWS 계정 사용하는 에는 Amazon Rekognition API에 대한 액세스 권한이 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 Amazon Rekognition에서 정의한 작업을 참조하세요.

비디오 스트림에서 얼굴을 인식하려면(AWS SDK)

1. IAM 서비스 역할을 아직 생성하지 않은 경우 생성하여 Kinesis 비디오 스트림 및 Kinesis 데이터 스트림에 대한 Amazon Rekognition Video 액세스 권한을 부여하세요. ARN을 기록합니다. 자세한 내용은 AmazonRekognitionServiceRole을 사용하여 스트림 액세스 권한 부여 단원을 참조하십시오.
2. 모음을 만들고, 사용한 모음 식별자를 적어둡니다.
3. 2단계에서 만든 모음에 검색하고자 하는 얼굴을 인덱싱합니다.
4. Kinesis 비디오 스트림을 만들고 스트림의 Amazon 리소스 이름(ARN)을 적어둡니다.
5. Kinesis 데이터 스트림을 생성합니다. 스트림 이름 앞에 AmazonRekognition을 추가하고 스트림의 ARN을 적어둡니다.

그 후 얼굴 검색 스트림 프로세서를 생성하고 선택한 스트림 프로세서 이름을 사용하여 스트림 프로세서를 시작할 수 있습니다.

> **Note**
>
> 미디어를 Kinesis 비디오 스트림으로 수집할 수 있는지 확인한 다음에 스트림 프로세서를 시작해야 합니다.

Amazon Rekognition Video로 비디오 스트리밍

Amazon Rekognition Video로 비디오를 스트리밍하려면 Amazon Kinesis Video Streams SDK를 사용하여 Kinesis 비디오 스트림을 생성하고 사용합니다. 이 `PutMedia` 작업은 Amazon Rekognition Video가 소비하는 Kinesis 비디오 스트림에 비디오 데이터 조각을 기록합니다. 각 비디오 데이터 조각의 길이는 일반적으로 2~10초이며 독립적인 비디오 프레임 시퀀스를 포함합니다. Amazon Rekognition Video는 H.264로 인코딩된 비디오를 지원하며, 이 비디오에는 세 가지 유형의 프레임(I, B, P)이 있을 수 있습니다. 자세한 내용은 Inter Frame을 참조하십시오. 조각의 첫 번째 프레임은 I 프레임이어야 합니다. I- 프레임은 다른 프레임과 별도로 디코딩될 수 있습니다.

비디오 데이터가 Kinesis 비디오 스트림에 도착하면 Kinesis Video Streams가 조각에 고유 번호를 할당합니다. 예제를 보려면 PutMedia API 예제를 참조하십시오.

- Matroska(MKV) 로 인코딩된 소스에서 스트리밍하는 경우 PutMedia 작업을 사용하여 소스 비디오를 생성한 Kinesis 비디오 스트림으로 스트리밍합니다. 자세한 내용은 PutMedia API 예제를 참조하십시오.
- 디바이스 카메라에서 스트리밍하는 경우 GStreamer 플러그인을 사용한 스트리밍 섹션을 참조하세요.

Amazon Rekognition Video에 리소스에 대한 액세스 권한 부여

AWS Identity and Access Management (IAM) 서비스 역할을 사용하여 Amazon Rekognition Video에 Kinesis 비디오 스트림에 대한 읽기 액세스 권한을 부여합니다. 얼굴 검색 스트림 프로세서를 사용하는 경우 IAM 서비스 역할을 사용하여 Amazon Rekognition Video에 Kinesis 데이터 스트림에 대한 쓰기 액세스 권한을 부여합니다. 보안 모니터링 스트림 프로세서를 사용하는 경우, IAM 역할을 사용하여 Amazon Rekognition Video에 Amazon S3 버킷 및 Amazon SNS 주제에 대한 액세스 권한을 부여합니다.

얼굴 검색 스트림 프로세서 액세스 권한 부여

Amazon Rekognition Video에서 개별 Kinesis 비디오 스트림 및 Kinesis 데이터 스트림에 액세스할 수 있도록 허용하는 권한 정책을 생성할 수 있습니다.

Amazon Rekognition Video에 얼굴 검색 스트림 프로세서를 위한 액세스 권한을 부여하려면

1. IAM JSON 정책 편집기로 새 권한 정책을 생성하고 다음 정책을 사용합니다. `video-arn`을 원하는 Kinesis 비디오 스트림의 ARN으로 교체하세요. 얼굴 검색 스트림 프로세서를 사용하는 경우 `data-arn`을 원하는 Kinesis 데이터 스트림의 ARN으로 교체합니다.
2. IAM 서비스 역할을 생성하거나 기존 IAM 서비스 역할을 업데이트합니다. 다음 정보를 사용하여 IAM 서비스 역할을 생성하세요.
   1. 서비스 이름으로 Rekognition을 선택합니다.
   2. 서비스 역할 사용 사례로 [Rekognition]을 선택합니다.
   3. 1단계에 만든 권한 정책을 연결합니다.
3. 서비스 역할의 ARN을 기록합니다. 이것은 비디오 분석 작업을 시작할 때 필요합니다.

AmazonRekognitionServiceRole을 사용하여 스트림 액세스 권한 부여

Kinesis 비디오 스트림 및 데이터 스트림에 대한 액세스를 설정하는 대체 옵션으로 `AmazonRekognitionServiceRole` 권한 정책을 사용할 수 있습니다. IAM은 Rekognition 서비스 역할 사용 사례를 제공합니다. 이 역할을 `AmazonRekognitionServiceRole` 권한 정책과 함께 사용할 경우 여러 Kinesis 데이터 스트림에 쓰기가 가능하며 모든 Kinesis 비디오 스트림에서 읽기를 할 수 있습니다. Amazon Rekognition Video에 여러 Kinesis 데이터 스트림에 대한 쓰기 권한을 부여하려면 Kinesis 데이터 스트림의 이름 앞에 AmazonRekognition을 추가하면 됩니다(예: `AmazonRekognitionMyDataStreamName`).

Amazon Rekognition Video에 Kinesis 비디오 스트림 및 Kinesis 데이터 스트림에 대한 액세스 권한을 부여하려면

1. IAM 서비스 역할을 생성합니다. 다음 정보를 사용하여 IAM 서비스 역할을 생성하세요.
    1. 서비스 이름으로 Rekognition을 선택합니다.
    2. 서비스 역할 사용 사례로 [Rekognition]을 선택합니다.
    3. AmazonRekognitionServiceRole 권한 정책을 선택하세요. 이는 Amazon Rekognition Video에 AmazonRekognition 접두사가 붙은 Kinesis 데이터 스트림에 대한 쓰기 액세스 권한 및 모든 Kinesis 비디오 스트림에 대한 읽기 액세스 권한을 부여하게 됩니다.
2. AWS 계정 의 보안을 보장하려면 Rekognition의 액세스 범위를 사용 중인 리소스로만 제한합니다. 이는 IAM 서비스 역할에 신뢰 정책을 추가하는 것으로 수행할 수 있습니다. 이렇게 하는 방법에 대한 정보는 교차 서비스 혼동된 대리인 방지 단원을 참조하십시오.
3. 서비스 역할의 Amazon Resource Name(ARN)을 적어둡니다. 이것은 비디오 분석 작업을 시작할 때 필요합니다.

스트리밍 비디오에서 얼굴 검색

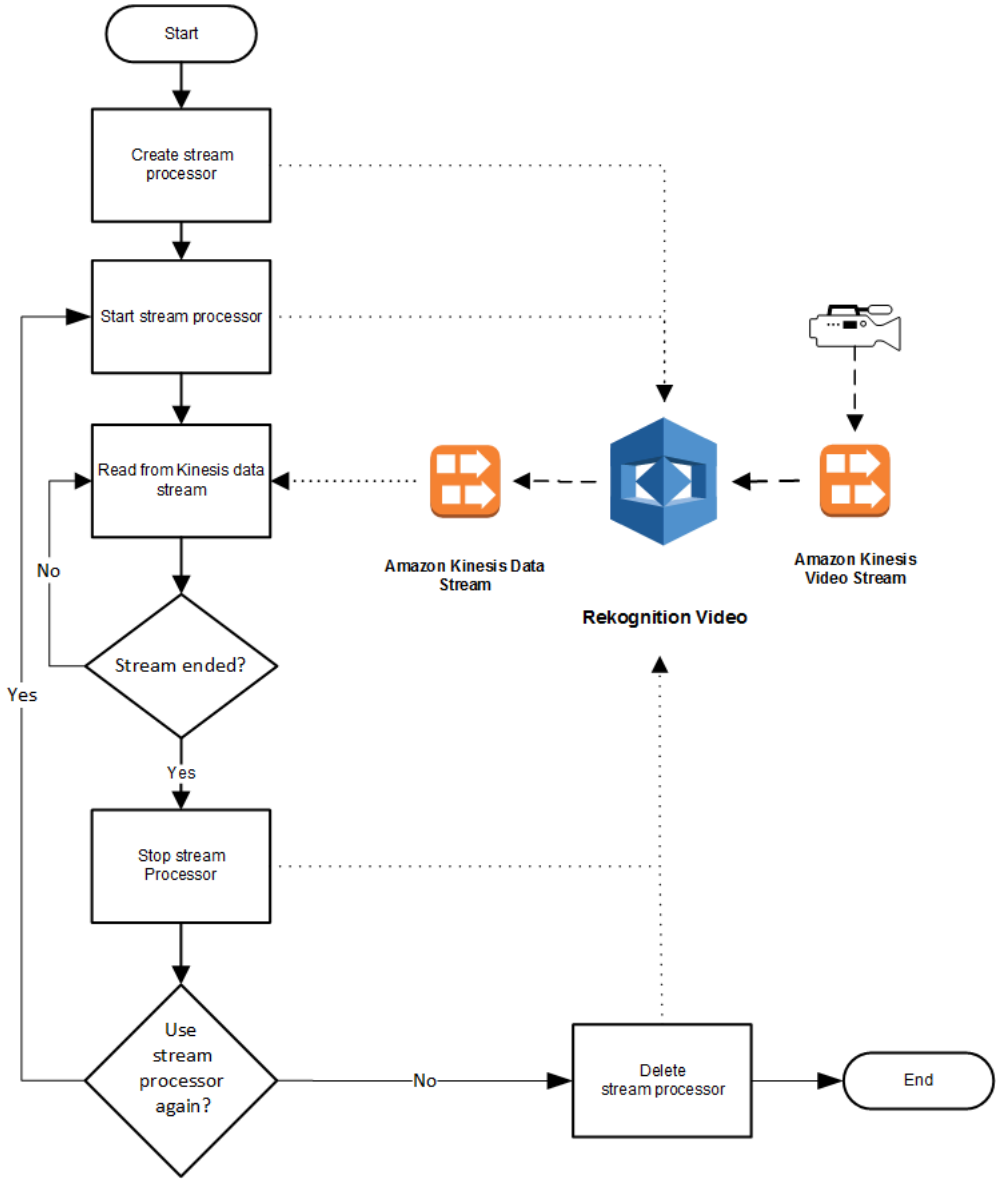
Amazon Rekognition Video는 스트리밍 비디오에서 감지한 얼굴과 일치하는 얼굴을 컬렉션에서 찾아낼 수 있습니다. 모음에 대한 자세한 내용은 컬렉션에서 얼굴 검색 단원을 참조하십시오.

주제

- Amazon Rekognition Video에 얼굴 검색 스트림 프로세서 생성
- Amazon Rekognition Video 얼굴 검색 스트림 프로세서 시작
- 얼굴 검색을 위한 스트림 프로세서 사용(Java V2 예제)
- 얼굴 검색을 위한 스트림 프로세서 사용(Java V1 예제)

- 스트리밍 비디오 분석 결과 읽기
- Kinesis Video Streams를 사용하여 로컬에서 Rekognition 결과 표시
- Kinesis 얼굴 인식 JSON 프레임 레코드 이해

다음 다이어그램은 Amazon Rekognition Video가 스트리밍 비디오에서 어떻게 얼굴을 감지하고 인식하는지 보여줍니다.



Amazon Rekognition Video에 얼굴 검색 스트림 프로세서 생성

스트리밍 비디오를 분석하기 전에 Amazon Rekognition Video 스트림 프로세서를 만듭니다 (CreateStreamProcessor). 스트림 프로세서에는 Kinesis 데이터 스트림과 Kinesis 비디오 스트림에 대한 정보가 들어 있습니다. 또한 입력된 스트리밍 비디오에서 인식하고자 하는 얼굴

이 포함된 모음의 식별자도 포함되어 있습니다. 스트림 프로세서의 이름도 지정합니다. 다음은 CreateStreamProcessor 요청에 대한 JSON 예제입니다.

```
{
	"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
		}
	}
}
```

다음은 CreateStreamProcessor의 응답 예제입니다.

```
{
	“StreamProcessorArn”: “arn:aws:rekognition:us-
east-1:nnnnnnnnnnnn:streamprocessor/streamProcessorForCam”
}
```

Amazon Rekognition Video 얼굴 검색 스트림 프로세서 시작

CreateStreamProcessor에서 지정한 스트림 프로세서 이름으로 StartStreamProcessor를 직접 호출하여 스트리밍 비디오 분석을 시작합니다. 다음은 StartStreamProcessor 요청에 대한 JSON 예제입니다.

```
{
	"Name": "streamProcessorForCam"
}
```

스트림 프로세서가 성공적으로 시작되면 JSON 본문이 비어 있는 상태로 HTTP 200 응답이 반환됩니다.

얼굴 검색을 위한 스트림 프로세서 사용(Java V2 예제)

다음 예제 코드는 Java용 AWS SDK 버전 2를 사용하여 CreateStreamProcessor 및 StartStreamProcessor와 같은 다양한 스트림 프로세서 작업을 호출하는 방법을 보여줍니다.

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorRequest;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorResponse;
import software.amazon.awssdk.services.rekognition.model.FaceSearchSettings;
import software.amazon.awssdk.services.rekognition.model.KinesisDataStream;
import software.amazon.awssdk.services.rekognition.model.KinesisVideoStream;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsRequest;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.StreamProcessor;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorInput;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorSettings;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorOutput;
import software.amazon.awssdk.services.rekognition.model.StartStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorResponse;

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

```
                Usage:    <role> <kinInputStream> <kinOutputStream> <collectionName>
<StreamProcessorName>

                Where:
                   role - The ARN of the AWS Identity and Access Management (IAM) role
to use. \s
                   kinInputStream - The ARN of the Kinesis video stream.\s
                   kinOutputStream - The ARN of the Kinesis data stream.\s
                   collectionName - The name of the collection to use that contains
content. \s
                   StreamProcessorName - The name of the Stream Processor. \s
                """;

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

       String role = args[0];
       String kinInputStream = args[1];
       String kinOutputStream = args[2];
       String collectionName = args[3];
       String streamProcessorName = args[4];

       Region region = Region.US_EAST_1;
       RekognitionClient rekClient = RekognitionClient.builder()
               .region(region)
               .build();

       processCollection(rekClient, streamProcessorName, kinInputStream,
kinOutputStream, collectionName,
               role);
       startSpecificStreamProcessor(rekClient, streamProcessorName);
       listStreamProcessors(rekClient);
       describeStreamProcessor(rekClient, streamProcessorName);
       deleteSpecificStreamProcessor(rekClient, streamProcessorName);
   }

   public static void listStreamProcessors(RekognitionClient rekClient) {
       ListStreamProcessorsRequest request = ListStreamProcessorsRequest.builder()
               .maxResults(15)
               .build();
```

```
        ListStreamProcessorsResponse listStreamProcessorsResult =
rekClient.listStreamProcessors(request);
        for (StreamProcessor streamProcessor :
listStreamProcessorsResult.streamProcessors()) {
            System.out.println("StreamProcessor name - " + streamProcessor.name());
            System.out.println("Status - " + streamProcessor.status());
        }
    }

    private static void describeStreamProcessor(RekognitionClient rekClient, String
StreamProcessorName) {
        DescribeStreamProcessorRequest streamProcessorRequest =
DescribeStreamProcessorRequest.builder()
                .name(StreamProcessorName)
                .build();

        DescribeStreamProcessorResponse describeStreamProcessorResult = rekClient
                .describeStreamProcessor(streamProcessorRequest);
        System.out.println("Arn - " +
describeStreamProcessorResult.streamProcessorArn());
        System.out.println("Input kinesisVideo stream - "
                + describeStreamProcessorResult.input().kinesisVideoStream().arn());
        System.out.println("Output kinesisData stream - "
                + describeStreamProcessorResult.output().kinesisDataStream().arn());
        System.out.println("RoleArn - " + describeStreamProcessorResult.roleArn());
        System.out.println(
                "CollectionId - "
                        +
describeStreamProcessorResult.settings().faceSearch().collectionId());
        System.out.println("Status - " + describeStreamProcessorResult.status());
        System.out.println("Status message - " +
describeStreamProcessorResult.statusMessage());
        System.out.println("Creation timestamp - " +
describeStreamProcessorResult.creationTimestamp());
        System.out.println("Last update timestamp - " +
describeStreamProcessorResult.lastUpdateTimestamp());
    }

    private static void startSpecificStreamProcessor(RekognitionClient rekClient,
String StreamProcessorName) {
        try {
            StartStreamProcessorRequest streamProcessorRequest =
StartStreamProcessorRequest.builder()
                    .name(StreamProcessorName)
```

```
                    .build();

            rekClient.startStreamProcessor(streamProcessorRequest);
            System.out.println("Stream Processor " + StreamProcessorName + "
started.");

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

    private static void processCollection(RekognitionClient rekClient, String
StreamProcessorName,
                                          String kinInputStream, String
kinOutputStream, String collectionName, String role) {
        try {
            KinesisVideoStream videoStream = KinesisVideoStream.builder()
                    .arn(kinInputStream)
                    .build();

            KinesisDataStream dataStream = KinesisDataStream.builder()
                    .arn(kinOutputStream)
                    .build();

            StreamProcessorOutput processorOutput = StreamProcessorOutput.builder()
                    .kinesisDataStream(dataStream)
                    .build();

            StreamProcessorInput processorInput = StreamProcessorInput.builder()
                    .kinesisVideoStream(videoStream)
                    .build();

            FaceSearchSettings searchSettings = FaceSearchSettings.builder()
                    .faceMatchThreshold(75f)
                    .collectionId(collectionName)
                    .build();

            StreamProcessorSettings processorSettings =
StreamProcessorSettings.builder()
                    .faceSearch(searchSettings)
                    .build();
```

```
            CreateStreamProcessorRequest processorRequest =
 CreateStreamProcessorRequest.builder()
                    .name(StreamProcessorName)
                    .input(processorInput)
                    .output(processorOutput)
                    .roleArn(role)
                    .settings(processorSettings)
                    .build();

            CreateStreamProcessorResponse response =
 rekClient.createStreamProcessor(processorRequest);
            System.out.println("The ARN for the newly create stream processor is "
                    + response.streamProcessorArn());

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

    private static void deleteSpecificStreamProcessor(RekognitionClient rekClient,
 String StreamProcessorName) {
        rekClient.stopStreamProcessor(a -> a.name(StreamProcessorName));
        rekClient.deleteStreamProcessor(a -> a.name(StreamProcessorName));
        System.out.println("Stream Processor " + StreamProcessorName + " deleted.");
    }
}
```

얼굴 검색을 위한 스트림 프로세서 사용(Java V1 예제)

다음 예제 코드는 Java V1을 사용하여 CreateStreamProcessor 및StartStreamProcessor와 같은 다양한 스트림 프로세서 작업을 직접 호출하는 방법을 보여줍니다. 이 예제에는 스트림 프로세서 작업을 직접 호출하는 메서드를 제공하는 스트림 프로세서 관리자 클래스(StreamManager)가 포함됩니다. 스타터 클래스(Starter)는 StreamManager 객체를 생성하고 다양한 작업을 호출합니다.

예제를 구성하려면:

1. Starter 클래스 멤버 필드의 값을 원하는 값으로 설정합니다.
2. Starter 클래스 함수 `main`에서 원하는 함수 호출을 주석 처리합니다.

Starter 클래스

```
//Copyright 2018 Amazon.com, Inc. or its 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 클래스

```
//Copyright 2018 Amazon.com, Inc. or its 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());
        }
    }
}
```

스트리밍 비디오 분석 결과 읽기

Amazon Kinesis Data Streams 클라이언트 라이브러리를 사용하여 Amazon Kinesis Data Streams 출력 스트림으로 전송되는 분석 결과를 사용할 수 있습니다. 자세한 내용은 Kinesis 데이터 스트림에서 데이터 읽기를 참조하세요. Amazon Rekognition Video는 분석된 각 프레임에 대한 JSON 프레임 레코드를 Kinesis 출력 스트림에 저장합니다. Amazon Rekognition Video는 Kinesis 비디오 스트림을 통해 전달되는 모든 프레임을 분석하지는 않습니다.

Kinesis 데이터 스트림으로 전송되는 프레임 레코드에는 프레임이 있는 Kinesis 비디오 스트림 조각, 조각 내에서 프레임의 위치, 프레임에서 인식되는 얼굴에 대한 정보가 포함되어 있습니다. 또한 스트림 프로세서의 상태 정보도 포함됩니다. 자세한 내용은 Kinesis 얼굴 인식 JSON 프레임 레코드 이해 단원을 참조하십시오.

Amazon Kinesis Video Streams 구문 분석 라이브러리에는 Amazon Rekognition Video 결과를 사용하고 이를 원본 Kinesis 비디오 스트림과 통합하는 예제 테스트가 포함되어 있습니다. 자세한 내용은 Kinesis Video Streams를 사용하여 로컬에서 Rekognition 결과 표시 단원을 참조하십시오.

Amazon Rekognition Video는 Amazon Rekognition Video 분석 정보를 Kinesis 데이터 스트림으로 스트리밍합니다. 다음은 단일 레코드에 대한 JSON 예제입니다.

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

JSON 예제에서 다음 사항에 유의하십시오.

- InputInformation - Amazon Rekognition Video로 비디오를 스트리밍하는 데 사용되는 Kinesis 비디오 스트림에 관한 정보입니다. 자세한 내용은 InputInformation 단원을 참조하십시오.

- StreamProcessorInformation - Amazon Rekognition Video 스트림 프로세서의 상태 정보입니다. `Status` 필드에 입력할 수 있는 값은 RUNNING뿐입니다. 자세한 내용은 StreamProcessorInformation 단원을 참조하십시오.
- FaceSearchResponse - 입력 컬렉션의 얼굴과 일치하는 스트리밍 비디오 속의 얼굴에 대한 정보가 들어 있습니다. FaceSearchResponse에는 분석한 비디오 프레임에서 감지된 얼굴, 즉 DetectedFace 객체가 포함되어 있습니다. `MatchedFaces` 배열에는 감지된 얼굴별로 입력 모음에서 발견된 일치하는 얼굴 객체(MatchedFace)와 유사성 점수가 나와 있습니다.

Kinesis 비디오 스트림을 Kinesis 데이터 스트림으로 매핑

Kinesis 비디오 스트림 프레임을 Kinesis 데이터 스트림으로 전송되는 분석 프레임으로 매핑하고자 하는 경우가 있을 수 있습니다. 예를 들어 스트리밍 동영상이 표시되는 도중에도 인식된 사람들의 얼굴 주위로 상자를 표시하는 것이 가능합니다. 경계 상자 좌표는 Kinesis 얼굴 인식 레코드에 포함되어 Kinesis 데이터 스트림으로 전송됩니다. 경계 상자를 정확하게 표시하려면 Kinesis 얼굴 인식 레코드와 함께 전송된 시간 정보를 원본 Kinesis 비디오 스트림의 해당 프레임과 매핑시켜야 합니다.

Kinesis 비디오 스트림을 Kinesis 데이터 스트림에 매핑할 때 사용하는 기법은 라이브 미디어(라이브 스트리밍 비디오 등)를 스트리밍하는지 또는 아카이브 미디어(저장된 비디오 등)를 스트리밍하는지에 따라 달라집니다.

라이브 미디어를 스트리밍하는 경우의 매핑

Kinesis 비디오 스트림 프레임을 Kinesis 데이터 스트림 프레임에 매핑하려면

1. PutMedia 작업의 입력 파라미터 `FragmentTimeCodeType`을 `RELATIVE`로 설정합니다.
2. `PutMedia`를 직접 호출하여 라이브 미디어를 Kinesis 비디오 스트림에 전송하세요.
3. Kinesis 데이터 스트림에서 Kinesis 얼굴 인식 레코드를 수신하면 KinesisVideo 필드의 `ProducerTimestamp` 값과 `FrameOffsetInSeconds` 값을 저장합니다.
4. `ProducerTimestamp` 필드 값과 `FrameOffsetInSeconds` 필드 값을 합산하여 Kinesis 비디오 스트림 프레임에 대응하는 타임스탬프를 계산합니다.

아카이브 미디어를 스트리밍하는 경우의 매핑

Kinesis 비디오 스트림 프레임을 Kinesis 데이터 스트림 프레임에 매핑하려면

1. PutMedia를 직접 호출하여 아카이브 미디어를 Kinesis 비디오 스트림으로 전달합니다.

2. PutMedia 작업 응답에서 Acknowledgement 객체를 수신하면 Payload 필드의 FragmentNumber 필드 값을 저장합니다. FragmentNumber는 MKV 클러스터의 조각 번호입니다.
3. Kinesis 데이터 스트림에서 Kinesis 얼굴 인식 레코드를 수신하면 KinesisVideo 필드의 FrameOffsetInSeconds 값을 저장합니다.
4. 2단계와 3단계에서 저장한 FrameOffsetInSeconds 및 FragmentNumber 값을 사용하여 매핑을 계산합니다. 여기에서 FrameOffsetInSeconds는 Amazon Kinesis 데이터 스트림으로 전송되는 특정 FragmentNumber에 대한 오프셋입니다. 임의의 조각 번호에 대한 비디오 프레임을 가져오는 방법에 대한 자세한 내용은 Amazon Kinesis Video Streams 아카이브 미디어를 참조하세요.

Kinesis Video Streams를 사용하여 로컬에서 Rekognition 결과 표시

KinesisVideo - Rekognition 예제에서 제공하는 Amazon Kinesis Video Streams 구문 분석 라이브러리의 예제 테스트를 사용하여 Amazon Kinesis Video Streams의 피드에 표시된 Amazon Rekognition Video의 결과를 확인할 수 있습니다. KinesisVideoRekognitionIntegrationExample는 감지된 얼굴 위에 경계 상자를 표시하고 JFrame을 통해 로컬에서 비디오를 렌더링합니다. 이 프로세스는 디바이스 카메라의 미디어 입력을 Kinesis 비디오 스트림에 성공적으로 연결하고 Amazon Rekognition 스트림 프로세서를 시작했다는 것을 가정한 것입니다. 자세한 내용은 GStreamer 플러그인을 사용한 스트리밍 단원을 참조하십시오.

1단계: Kinesis Video Streams 구문 분석 라이브러리 설치

디렉터리를 생성하고 GitHub 리포지토리를 다운로드하려면 다음 명령을 실행하세요.

```
$ git clone https://github.com/aws/amazon-kinesis-video-streams-parser-library.git
```

라이브러리 디렉터리로 이동한 후 다음 Maven 명령을 실행하여 클린 설치를 수행합니다.

```
$ mvn clean install
```

2단계: Kinesis Video Streams 및 Rekognition 통합 예제 테스트 구성

KinesisVideoRekognitionIntegrationExampleTest.java 파일을 엽니다. 클래스 헤더 바로 뒤에 있는 @Ignore를 제거합니다. Amazon Kinesis 및 Amazon Rekognition 리소스의 정보로 데이터 필드를 채웁니다. 자세한 내용은 Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정 단원을 참조하십시오. Kinesis 비디오 스트림으로 비디오를 스트리밍하는 경우 inputStream 파라미터를 제거하세요.

다음 코드 예제를 참조하세요.

```
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();
```

3단계: Kinesis Video Streams 및 Rekognition 통합 예제 테스트 실행

Kinesis 비디오 스트림으로 스트리밍하는 경우 Kinesis 비디오 스트림이 미디어 입력을 받고 있는지 확인하고 Amazon Rekognition Video 스트림 프로세서가 실행 중인 상태에서 스트림 분석을 시작하세요. 자세한 내용은 Amazon Rekognition Video 스트림 프로세서 작업 개요 단원을 참조하십시오. `KinesisVideoRekognitionIntegrationExampleTest` 클래스를 JUnit 테스트로 실행하세요. 잠시 후 Kinesis 비디오 스트림의 비디오 피드가 포함된 새 창이 열리고 감지된 얼굴 위에 경계 상자가 그려집니다.

> **Note**
>
> 이 예제에 사용된 컬렉션의 얼굴에 경계 상자 레이블이 유의미한 텍스트를 표시하기 위해서는 외부 이미지 ID(파일 이름)가 다음 형식으로 지정되어 있어야 합니다. PersonName1-Trusted, PersonName2-Intruder, PersonName3-Neutral 등. 레이블은 색상으로 구분할 수도 있으며 FaceType.java 파일에서 사용자 지정할 수도 있습니다.

Kinesis 얼굴 인식 JSON 프레임 레코드 이해

Amazon Rekognition Video를 사용하면 스트리밍 비디오에서 얼굴을 인식할 수 있습니다. Amazon Rekognition Video는 분석된 각 프레임마다 JSON 프레임 레코드를 Kinesis 데이터 스트림으로 출력합니다. Amazon Rekognition Video는 Kinesis 비디오 스트림을 통해 전달되는 모든 프레임을 분석하지는 않습니다.

JSON 프레임 레코드에는 입력 및 출력 스트림, 스트림 프로세서의 상태, 분석된 프레임에서 인식되는 얼굴에 대한 정보가 들어 있습니다. 이 단원에서는 JSON 프레임 레코드에 대한 참조 정보를 제시합니다.

다음은 Kinesis 데이터 스트림 레코드에 대한 JSON 구문입니다. 자세한 내용은 스트리밍 비디오 이벤트 작업 단원을 참조하십시오.

> **Note**
>
> Amazon Rekognition Video API는 입력 스트림의 얼굴을 얼굴 컬렉션과 비교하고 검색된 가장 근접한 일치 항목을 유사성 점수와 함께 반환하는 방식으로 작동합니다.

```
{
    "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 레코드

JSON 레코드에는 Amazon Rekognition Video에서 처리하는 프레임에 대한 정보가 포함되어 있습니다. 레코드에는 스트리밍 비디오, 분석된 프레임의 상태, 프레임에서 인식되는 얼굴에 대한 정보가 포함됩니다.

InputInformation

Amazon Rekognition Video로 비디오를 스트리밍하는 데 사용되는 Kinesis 비디오 스트림에 대한 정보입니다.

유형: InputInformation객체

StreamProcessorInformation

Amazon Rekognition Video 스트림 프로세서에 대한 정보. 여기에는 스트림 프로세서의 현재 상태에 대한 상태 정보가 포함됩니다.

유형: StreamProcessorInformation객체

FaceSearchResponse

스트리밍 비디오 프레임에서 감지된 얼굴과 입력 모음에서 발견된 일치하는 얼굴에 대한 정보입니다.

유형: FaceSearchResponse 객체 배열

InputInformation

Amazon Rekognition Video에서 사용하는 소스 비디오 스트림에 대한 정보입니다. 자세한 내용은 스트리밍 비디오 이벤트 작업 단원을 참조하십시오.

KinesisVideo

유형: KinesisVideo객체

KinesisVideo

소스 비디오를 Amazon Rekognition Video로 스트리밍하는 Kinesis 비디오 스트림에 대한 정보입니다. 자세한 내용은 스트리밍 비디오 이벤트 작업 단원을 참조하십시오.

StreamArn

Kinesis 비디오 스트림의 Amazon 리소스 이름(ARN)입니다.

유형: 문자열

FragmentNumber

이 레코드가 나타내는 프레임이 포함된 스트리밍 비디오 조각입니다.

유형: 문자열

ProducerTimestamp

조각의 생산자 측 Unix 타임스탬프입니다. 자세한 내용은 PutMedia 단원을 참조하십시오.

형식: 숫자

ServerTimestamp

조각의 서버 측 Unix 타임스탬프입니다. 자세한 내용은 PutMedia 단원을 참조하십시오.

형식: 숫자

FrameOffsetInSeconds

조각 내의 프레임 오프셋(초 단위)입니다.

형식: 숫자

StreamProcessorInformation

스트림 프로세서에 대한 상태 정보입니다.

상태

스트림 프로세서의 현재 상태입니다. 가능한 값은 RUNNING입니다.

유형: 문자열

FaceSearchResponse

스트리밍 비디오 프레임에서 감지된 얼굴과 모음에서 발견된 그것과 일치하는 얼굴에 대한 정보입니다. 컬렉션은 CreateStreamProcessor를 직접 호출하여 지정합니다. 자세한 내용은 스트리밍 비디오 이벤트 작업 단원을 참조하십시오.

DetectedFace

분석된 비디오 프레임에서 감지된 얼굴의 세부 정보입니다.

유형: DetectedFace객체

MatchedFaces

DetectedFace에서 감지된 얼굴과 일치하는 얼굴 모음의 세부 정보 배열입니다.

유형: MatchedFace 객체 배열

DetectedFace

스트리밍 비디오 프레임에서 감지된 얼굴에 대한 정보입니다. 입력 모음에 들어 있는 일치하는 얼굴은 MatchedFace 객체 필드에서 사용할 수 있습니다.

BoundingBox

분석된 비디오 프레임 내에서 감지된 얼굴의 경계 상자 좌표입니다. BoundingBox 객체는 이미지 분석에 사용되는 BoundingBox 객체와 동일한 속성을 가집니다.

유형: BoundingBox 객체

신뢰도

Amazon Rekognition Video에서 감지된 얼굴이 실제로 얼굴임을 신뢰할 수 있는 수준(1~100)입니다. 1은 가장 낮은 신뢰도, 100은 가장 높은 신뢰도입니다.

형식: 숫자

표식

얼굴 표식의 배열입니다.

유형: Landmark 객체 배열

포즈

피치, 롤 및 요로 판단되는 얼굴의 포즈를 나타냅니다.

유형: Pose 객체

화질

얼굴 이미지의 밝기와 선명도를 나타냅니다.

유형: ImageQuality 객체

MatchedFace

분석된 비디오 프레임에서 감지된 얼굴과 일치하는 얼굴에 대한 정보입니다.

얼굴

DetectedFace 객체의 얼굴과 일치하는, 입력 모음의 얼굴에 대한 얼굴 일치 정보입니다.

유형: Face 객체

유사성

얼굴 일치의 신뢰 수준(1~100)입니다. 1은 가장 낮은 신뢰도, 100은 가장 높은 신뢰도입니다.

형식: 숫자

GStreamer 플러그인을 사용한 스트리밍

Amazon Rekognition Video는 디바이스 카메라의 라이브 스트리밍 비디오를 분석할 수 있습니다. 디바이스 소스에서의 미디어 입력에 액세스하려면 GStreamer를 설치해야 합니다. GStreamer는 워크플로 파이프라인에서 미디어 소스와 처리 도구를 함께 연결하는 서드 파티 멀티미디어 프레임워크 소프트웨어입니다. Gstreamer용 Amazon Kinesis Video Streams Producer 플러그인도 설치해야 합니다. 이 프로세스에서는 Amazon Rekognition Video 및 Amazon Kinesis 리소스를 성공적으로 설정했다고 가정합니다. 자세한 내용은 Amazon Rekognition Video 및 Amazon Kinesis 리소스 설정 단원을 참조하십시오.

1단계: Gstreamer 설치

서드 파티 멀티미디어 플랫폼 소프트웨어인 Gstreamer를 다운로드하여 설치합니다. Homebrew(Gstreamer on Homebrew)와 같은 패키지 관리 소프트웨어를 사용하거나 Freedesktop 웹 사이트에서 직접 다운로드할 수 있습니다.

명령줄 터미널의 테스트 소스로 비디오 피드를 실행하여 Gstreamer가 성공적으로 설치되었는지 확인하세요.

```
$ gst-launch-1.0 videotestsrc ! autovideosink
```

2단계: Kinesis Video Streams Producer 플러그인 설치

이 섹션에서는 Amazon Kinesis Video Streams Producer 라이브러리를 다운로드하고 Kinesis Video Streams Gstreamer 플러그인을 설치합니다.

디렉터리를 생성하고 GitHub 리포지토리에서 소스 코드를 복제합니다. `--recursive` 파라미터가 반드시 포함되어야 합니다.

```
$ git clone --recursive https://github.com/awslabs/amazon-kinesis-video-streams-producer-sdk-cpp.git
```

라이브러리에서 제공하는 지침에 따라 프로젝트를 구성하고 빌드하세요. 운영 체제에 맞는 플랫폼별 명령을 사용해야 합니다. Kinesis Video Streams Gstreamer 플러그인 설치를 위해 `cmake`를 실행할 때 `-DBUILD_GSTREAMER_PLUGIN=ON` 파라미터를 사용하세요. 이 프로젝트에는 설치에 포함된 GCC 또는 Clang, Curl, Openssl 및 Log4cplus와 같은 추가 패키지가 필요합니다. 패키지 누락으로 인해 빌드가 실패하는 경우 해당 패키지가 설치되어 있고 PATH에 위치해 있는지 확인하세요. 빌드 중 "can't run C compiled program" 오류가 발생한 경우, 빌드 명령을 다시 실행하세요. 올바른 C 컴파일러를 찾을 수 없는 경우가 있습니다.

다음 명령을 실행하여 Kinesis Video Streams 플러그인의 설치를 확인하세요.

```
$ gst-inspect-1.0 kvssink
```

팩토리 및 플러그인 세부 정보와 같은 다음 정보가 표시되어야 합니다.

```
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/

  ...
```

3단계: Kinesis Video Streams 플러그인을 사용하여 Gstreamer 실행

디바이스 카메라에서 Kinesis Video Streams로 스트리밍을 시작하기 전에 미디어 소스를 Kinesis Video Streams에 적합한 코덱으로 변환해야 할 수 있습니다. 현재 컴퓨터에 연결된 디바이스의 사양과 포맷 기능을 확인하려면 다음 명령을 실행하세요.

```
$ gst-device-monitor-1.0
```

스트리밍을 시작하려면 다음 샘플 명령을 사용하여 Gstreamer를 실행하고 보안 인증 정보 및 Amazon Kinesis Video Streams 정보를 추가합니다. Amazon Rekognition에 Kinesis 스트림에 대한 액세스 부여 중에 생성한 IAM 서비스 역할에 맞는 액세스 키와 리전을 사용해야 합니다. 액세스 키에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하세요. 또한 비디오 형식 인수 파라미터를 디바이스에서 사용 가능하다면 사용량에 따라 필요한 대로 조정할 수 있습니다.

```
$ 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"
```

시작 명령에 대한 자세한 내용은 GStreamer 시작 명령 예제를 참조하세요.

> **Note**
>
> 시작 명령이 비협상 오류로 종료되는 경우 디바이스 모니터의 출력값을 확인하고 `videoconvert` 파라미터 값이 디바이스의 설정에 유효한지 확인하세요.

몇 초 후에 Kinesis 비디오 스트림에서 디바이스 카메라의 비디오 피드를 볼 수 있습니다. Amazon Rekognition으로 얼굴 감지 및 일치를 시작하려면 Amazon Rekognition Video 스트림 프로세서를 시작하세요. 자세한 내용은 Amazon Rekognition Video 스트림 프로세서 작업 개요 단원을 참조하십시오.

스트리밍 비디오 문제 해결

이 주제에서는 Amazon Rekognition Video를 스트리밍 비디오에 사용할 경우의 문제 해결 정보를 제공합니다.

주제

- 스트림 프로세서가 성공적으로 생성되었는지 모르겠습니다.
- 스트림 프로세서를 올바로 구성했는지 모르겠습니다
- 스트림 프로세서가 결과를 반환하지 않습니다
- 스트림 프로세서의 상태가 FAILED입니다
- 스트림 프로세서가 예상 결과를 반환하지 않습니다

스트림 프로세서가 성공적으로 생성되었는지 모르겠습니다.

다음 AWS CLI 명령을 사용하여 스트림 프로세서 목록과 현재 상태를 가져옵니다.

```
aws rekognition list-stream-processors
```

다음 AWS CLI 명령을 사용하여 추가 세부 정보를 얻을 수 있습니다. `stream-processor-name`을 요청된 스트림 프로세서 이름으로 바꿉니다.

```
aws rekognition describe-stream-processor --name stream-processor-name
```

스트림 프로세서를 올바로 구성했는지 모르겠습니다

코드가 Amazon Rekognition Video에서 분석 결과를 출력하고 있지 않다면 스트림 프로세서가 올바로 구성되지 않았을 수 있습니다. 다음을 수행하여 스트림 프로세서가 올바로 구성되었고 결과를 생성할 수 있는지 확인하십시오.

솔루션이 올바로 구성되었는지 확인하려면

1. 다음 명령을 실행하여 스트림 프로세서가 실행 중 상태인지 확인합니다. `stream-processor-name`을 해당 스트림 프로세서의 이름으로 변경합니다. `Status` 값이 `RUNNING`이라면 스트림 프로세서가 실행 중인 것입니다. 상태가 `RUNNING`이고 결과를 얻지 못하는 경우, 스트림 프로세서가 결과를 반환하지 않습니다 단원을 참조하십시오. 상태가 `FAILED`인 경우, 스트림 프로세서의 상태가 FAILED입니다 단원을 참조하십시오.

   ```
   aws rekognition describe-stream-processor --name stream-processor-name
   ```

2. 스트림 프로세서가 실행 중이라면 다음 Bash 또는 PowerShell 명령을 실행하여 출력 Kinesis 데이터 스트림에서 데이터를 읽습니다.

Bash

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000
 --shard-iterator-type TRIM_HORIZON --stream-name kinesis-data-stream-name --query
 'ShardIterator')
                        aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

PowerShell

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-
id shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name kinesis-
data-stream-name).split('"')[4])
```

3. Base64 Decode 웹사이트에서 Decode 도구를 사용하여 출력을 사람이 읽을 수 있는 문자열로 디코딩합니다. 자세한 내용은 3단계: 레코드 가져오기를 참조하십시오.
4. 명령이 작동하고 Kinesis 데이터 스트림에서 얼굴 감지 결과가 표시된다면 솔루션이 올바로 구성된 것입니다. 명령이 실패하면 다른 문제 해결 제안을 확인하고 Amazon Rekognition Video에 리소스에 대한 액세스 권한 부여 단원을 참조하십시오.

또는 "kinesis-process-record" AWS Lambda 블루프린트를 사용하여 Kinesis 데이터 스트림의 메시지를 CloudWatch에 로깅하여 지속적으로 시각화할 수 있습니다. 이로 인해 AWS Lambda 및 CloudWatch에 대한 추가 비용이 발생합니다.

스트림 프로세서가 결과를 반환하지 않습니다

스트림 프로세서가 결과를 반환하지 않는 이유에는 몇 가지가 있을 수 있습니다.

이유 1: 스트림 프로세서가 올바로 구성되지 않은 경우

스트림 프로세서가 올바로 구성되지 않았을 수 있습니다. 자세한 내용은 스트림 프로세서를 올바로 구성했는지 모르겠습니다 단원을 참조하십시오.

이유 2: 스트림 프로세서 상태가 RUNNING이 아닌 경우

스트림 프로세서 상태 문제를 해결하려면

1. 다음 AWS CLI 명령을 사용하여 스트림 프로세서의 상태를 확인합니다.

```
aws rekognition describe-stream-processor --name stream-processor-name
```

2. Status 값이 STOPPED라면 다음 명령을 사용하여 스트림 프로세서를 시작합니다.

```
aws rekognition start-stream-processor --name stream-processor-name
```

3. Status 값이 FAILED라면 스트림 프로세서의 상태가 FAILED입니다 단원을 참조하십시오.
4. Status 값이 STARTING이라면 2분 간 기다린 후 1단계를 반복하여 상태를 확인합니다. 상태 값이 여전히 STARTING이라면 다음을 수행합니다.

   a. 다음 명령을 사용하여 스트림 프로세서를 삭제합니다.

   ```
   aws rekognition delete-stream-processor --name stream-processor-name
   ```

   b. 동일한 구성의 새 스트림 프로세서를 생성합니다. 자세한 내용은 스트리밍 비디오 이벤트 작업 단원을 참조하십시오.

   c. 여전히 문제가 있는 경우 AWS Support에 문의하세요.
5. Status 값이 RUNNING라면 원인 3: Kinesis 비디오 스트림에 활성 데이터가 없는 경우 단원을 참조하십시오.

원인 3: Kinesis 비디오 스트림에 활성 데이터가 없는 경우

Kinesis 비디오 스트림에 활성 데이터가 있는지 확인하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/kinesisvideo/ Amazon Kinesis Video Streams 콘솔을 엽니다.
2. Amazon Rekognition 스트림 프로세서에 입력되는 Kinesis 비디오 스트림을 선택합니다.
3. 미리 보기에 No data on stream이 표시되는 경우, Amazon Rekognition Video의 입력 스트림에 처리할 데이터가 없는 것입니다.

Kinesis Video Streams를 사용한 비디오 제작에 대한 자세한 내용은 Kinesis Video Streams Producer 라이브러리를 참조하세요.

스트림 프로세서의 상태가 FAILED입니다

다음 AWS CLI 명령을 사용하여 스트림 프로세서의 상태를 확인할 수 있습니다.

```
aws rekognition describe-stream-processor --name stream-processor-name
```

상태 값이 FAILED인 경우, 다음 오류 메시지의 문제 해결 정보를 확인하십시오.

오류: "Access denied to Role"

스트림 프로세서가 사용하는 IAM 역할이 존재하지 않거나 Amazon Rekognition Video에 해당 역할을 맡을 권한이 없습니다.

IAM 역할에 대한 액세스 문제를 해결하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택하고 역할이 존재하는지 확인합니다.
3. 역할이 존재하는 경우, 역할에 AmazonRekognitionServiceRole 권한 정책이 있는지 확인합니다.
4. 역할이 존재하지 않거나 적절한 권한을 가지고 있지 않다면 Amazon Rekognition Video에 리소스에 대한 액세스 권한 부여 단원을 참조하십시오.
5. 다음 AWS CLI 명령을 사용하여 스트림 프로세서를 시작합니다.

```
aws rekognition start-stream-processor --name stream-processor-name
```

오류: "Kinesis Video에 대한 액세스 거부됨 또는 Kinesis Data에 대한 액세스 거부됨"

해당 역할에 `GetMedia` 및 `GetDataEndpoint` Kinesis Video Streams API 작업에 대한 액세스 권한이 없습니다. `PutRecord` 및 `PutRecords` Kinesis Data Streams API 작업에 대한 액세스 권한도 없을 수 있습니다.

API 권한 문제를 해결하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 역할을 열고 다음 권한 정책이 연결되어 있는지 확인합니다.
3. 권한 중에 없는 것이 있다면 정책을 업데이트합니다. 자세한 내용은 Amazon Rekognition Video에 리소스에 대한 액세스 권한 부여 단원을 참조하십시오.

오류: "Stream ***input-video-stream-name*** doesn't exist"

스트림 프로세서로 들어가는 Kinesis 비디오 스트림 입력이 존재하지 않거나 올바로 구성되지 않았습니다.

Kinesis 비디오 스트림 문제를 해결하려면

1. 다음 명령을 사용하여 스트림이 존재하는지 확인합니다.

```
aws kinesisvideo list-streams
```

2. 스트림이 존재하는 경우, 다음을 확인합니다.

   - Amazon 리소스 이름(ARN)이 스트림 프로세서 입력 스트림의 ARN과 동일합니다.
   - Kinesis 비디오 스트림은 스트림 프로세서와 동일한 리전에 있어야 합니다.

   스트림 프로세서가 올바르게 구성되지 않은 경우 다음 AWS CLI 명령을 사용하여 삭제합니다.

```
aws rekognition delete-stream-processor --name stream-processor-name
```

3. 의도한 Kinesis 비디오 스트림으로 새 스트림 프로세서를 생성합니다. 자세한 내용은 Amazon Rekognition Video에 얼굴 검색 스트림 프로세서 생성 단원을 참조하십시오.

오류: "Collection not found"

스트림 프로세서가 얼굴 일치에 사용하는 Amazon Rekognition 컬렉션이 존재하지 않거나 잘못된 컬렉션이 사용되고 있습니다.

모음을 확인하려면

1. 다음 AWS CLI 명령을 사용하여 필요한 컬렉션이 존재하는지 확인합니다. 스트림 프로세서를 실행하는 AWS 리전`region`으로 변경합니다.

```
aws rekognition list-collections --region region
```

   필요한 모음이 존재하지 않는 경우, 새 모음을 생성하고 얼굴 정보를 추가합니다. 자세한 내용은 컬렉션에서 얼굴 검색 단원을 참조하십시오.

2. CreateStreamProcessor 직접 호출에서 `CollectionId` 입력 파라미터의 값이 올바른지 확인합니다.

3. 다음 AWS CLI 명령을 사용하여 스트림 프로세서를 시작합니다.

```
aws rekognition start-stream-processor --name stream-processor-name
```

오류: "Stream ***output-kinesis-data-stream-name*** under account ***account-id*** not found"

스트림 프로세서에서 사용하는 출력 Kinesis 데이터 스트림 AWS 계정 이에 없거나 스트림 프로세서와 동일한 AWS 리전에 없습니다.

Kinesis 데이터 스트림 문제를 해결하려면

1. 다음 AWS CLI 명령을 사용하여 Kinesis 데이터 스트림이 존재하는지 확인합니다. 스트림 프로세서를 사용하는 AWS 리전`region`으로 변경합니다.

   ```
   aws kinesis list-streams --region region
   ```

2. Kinesis 데이터 스트림이 존재하는 경우, Kinesis 데이터 스트림 이름이 스트림 프로세서가 사용하는 출력 스트림의 이름과 동일한지 확인합니다.
3. Kinesis 데이터 스트림이 없는 경우 다른 AWS 리전에 존재할 수 있습니다. Kinesis 데이터 스트림은 스트림 프로세서와 동일한 리전에 있어야 합니다.
4. 필요하다면 새 Kinesis 데이터 스트림을 만듭니다.

   a. 스트림 프로세서가 사용하는 이름과 동일한 이름으로 Kinesis 데이터 스트림을 생성합니다. 자세한 내용은 1단계: 데이터 스트림 생성을 참조하십시오.

   b. 다음 AWS CLI 명령을 사용하여 스트림 프로세서를 시작합니다.

   ```
   aws rekognition start-stream-processor --name stream-processor-name
   ```

스트림 프로세서가 예상 결과를 반환하지 않습니다

스트림 프로세서가 예상 얼굴 일치를 반환하지 않는 경우, 다음 정보를 사용하십시오.

- 컬렉션에서 얼굴 검색
- 카메라 설정 권장 사항(스트리밍 비디오)

인물 경로 추적

> **Note**
>
> 지원 종료 알림: 2025년 10월 31일에 AWS는 Amazon Rekognition 인물 경로에 대한 지원을 중단합니다. 2025년 10월 31일 이후에는 Rekognition 인물 경로 기능을 더 이상 사용할 수 없습니다. 자세한 내용은 이 블로그 게시물을 참조하세요.

Amazon Rekognition Video는 비디오 속 인물이 선택하는 경로의 트랙을 생성하고 다음과 같은 정보를 제공할 수 있습니다.

- 비디오 프레임에 있는 사람의 경로가 추적될 때의 위치입니다.
- 왼쪽 눈의 위치와 같은 얼굴 표식(감지된 경우).

Amazon Rekognition Video의 저장된 비디오 속 인물 경로 추적은 비동기식 작업입니다. 비디오 속 인물의 경로 추적을 시작하려면 StartPersonTracking을 직접 호출하세요. Amazon Rekognition Video는 비디오 분석의 완료 상태를 Amazon Simple Notification Service 주제에 게시합니다. 비디오 분석이 성공적으로 완료되면 GetPersonTracking을 직접 호출하여 비디오 분석 결과를 가져오세요. Amazon Rekognition Video API 작업 직접 호출에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 섹션을 참조하세요.

다음 절차는 Amazon S3 버킷에 저장된 비디오를 통해 사람의 경로를 추적하는 방법을 보여줍니다. 이 예제는 Amazon Simple Queue Service 대기열을 사용하여 비디오 분석 요청의 완료 상태를 가져오는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)의 코드를 확장합니다.

Amazon S3 버킷에 저장된 비디오에서 인물을 감지하려면(SDK)

1. Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)을 수행합니다.
2. 1단계에서 만든 클래스 `VideoDetect`에 다음 코드를 추가합니다.

   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);

        }
```

main 함수에서 다음 줄을 바꿉니다.

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

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

다음으로 바꿉니다.

```
        StartPersonDetection(amzn-s3-demo-bucket, video);

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

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져왔습니다. 전체 예제는 여기에서 확인하세요.

```
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
```

main 함수에서 다음 줄을 바꿉니다.

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

다음으로 바꿉니다.

```
    analyzer.StartPersonPathing()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetPersonPathingResults()
```

CLI

다음 AWS CLI 명령어를 실행하여 비디오에서 인물의 경로 추적을 시작합니다.

```
aws rekognition start-person-tracking --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-ARN","RoleArn":"role-ARN"}" \
--region region-name --profile profile-name
```

다음 값을 업데이트합니다.

- amzn-s3-demo-bucket 및 video-name을 2단계에서 지정한 Amazon S3 버킷 이름과 파일 이름으로 변경합니다.
- region-name을 사용 중인 AWS 리전으로 변경합니다.
- Rekognition 세션을 생성하는 라인에서 profile-name의 값을 개발자 프로필의 이름으로 대체합니다.
- topic-ARN을 Amazon Rekognition Video 구성의 3단계에서 생성한 Amazon SNS 주제의 ARN으로 변경합니다.
- role-ARN을 Amazon Rekognition Video 구성의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 변경합니다.

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예시를 보려면 다음을 참조하세요.

```
aws rekognition start-person-tracking --video "{\"S3Object\":{\"Bucket\":\"amzn-
s3-demo-bucket\",\"Name\":\"video-name\"}}"
--notification-channel "{\"SNSTopicArn\":\"topic-ARN\",\"RoleArn\":\"role-ARN
\"}" \
--region region-name --profile profile-name
```

진행 중인 코드 예제를 실행한 후 반환된 `jobID`를 복사하여 다음 `GetPersonTracking` 명령에 제공하여 결과를 가져오고,`job-id-number`를 이전에 받은 `jobID`로 바꾸세요.

```
aws rekognition get-person-tracking --job-id job-id-number
```

> **Note**
>
> Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 이외에 비디오 예제를 이미 실행한 경우, 바꿀 코드가 다를 수 있습니다.

3. 코드를 실행합니다. 추적된 사람에 대한 고유 식별자는 인물의 경로가 추적된 시간(초)과 함께 표시됩니다.

GetPersonTracking 작업 응답

`GetPersonTracking`은 비디오에서 감지된 인물에 대한 세부 정보와 그들의 경로가 추적된 시점을 포함하는 PersonDetection 객체의 배열인 `Persons`를 반환합니다.

`Persons` 입력 파라미터를 사용하여 `SortBy`를 정렬할 수 있습니다. 비디오에서 사람들의 경로를 추적하는 시간을 기준으로 요소를 정렬하려면 `TIMESTAMP`를 지정하십시오. 비디오에서 추적한 인물을 기준으로 정렬하려면 `INDEX`를 지정하십시오. 인물에 대한 각 결과 세트 내에서 경로 추적의 정확도에 대한 내림차순 신뢰도에 따라 요소가 정렬됩니다. 기본적으로 `Persons`은 `TIMESTAMP`를 기준으로 정

렬되어 반환됩니다. 다음 예제는 GetPersonDetection의 JSON 응답입니다. 결과는 비디오가 시작된 이후부터 비디오에 인물의 경로가 추적된 시간(밀리초)에 따라 정렬됩니다. 응답에서 다음에 유의하십시오.

- 인물 정보 - PersonDetection 배열 요소에는 감지된 인물에 대한 정보가 포함되어 있습니다. 예를 들어 인물이 감지된 시간(Timestamp), 감지 당시 비디오 프레임에 있는 인물의 위치(BoundingBox), 인물 감지의 정확성에 대한 Amazon Rekognition Video의 신뢰도(Confidence)입니다.

  사람의 경로가 추적되는 모든 타임스탬프에서 얼굴 특징은 반환되지 않습니다. 또한 상황에 따라 추적한 인물의 몸이 보이지 않을 수 있으며 이 경우 얼굴 위치만 반환됩니다.
- 페이징 정보 - 이 예제는 인물 감지 정보의 페이지 하나를 보여줍니다. MaxResults의 GetPersonTracking 입력 파라미터에 반환될 사람 요소의 수를 지정할 수 있습니다. MaxResults 보다 많은 결과가 존재할 경우 GetPersonTracking은 결과의 다음 페이지를 가져올 때 사용되는 토큰(NextToken)을 반환합니다. 자세한 내용은 Amazon Rekognition Video 분석 결과 가져오기 섹션을 참조하세요.
- 인덱스 - 비디오 전체에서 인물을 식별하기 위한 고유 식별자입니다.
- 비디오 정보 - 응답에는GetPersonDetection에서 반환된 정보의 각 페이지에 있는 비디오 형식(VideoMetadata)에 관한 정보가 포함되어 있습니다.


```
{
    "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
    }
}
```

개인 보호 장비 감지

Amazon Rekognition은 이미지 속 사람이 착용한 PPE(개인 보호 장비)를 감지할 수 있습니다. 이 정보를 사용하여 작업장 안전 관행을 개선할 수 있습니다. 예를 들어 PPE 감지 기능을 사용하면 건설 현장의 근로자가 머리 덮개를 착용하고 있는지 또는 의료 종사자가 얼굴 덮개 및 손 덮개를 착용하고 있는지 확인할 수 있습니다. 다음 이미지는 탐지할 수 있는 PPE 유형 중 일부를 보여줍니다.

이미지에서 PPE를 탐지하려면 DetectProtectiveEquipment API를 직접 호출하고 입력 이미지를 전달합니다. 응답은 다음을 포함하는 JSON 구조입니다.

- 이미지에서 감지된 사람
- PPE를 착용한 신체 부위(얼굴, 머리, 왼손, 오른손)
- 신체 부위에서 탐지된 PPE의 종류(얼굴 덮개, 손 덮개, 머리 덮개)
- 탐지된 PPE 항목에서 PPE가 적절한 신체 부위를 덮고 있는지 여부를 나타내는 지표

이미지에서 탐지된 PPE 항목 및 인물의 위치에 대해서는 경계 상자가 반환됩니다.

이미지에서 탐지된 PPE 항목 및 인물에 대한 요약을 요청할 수도 있습니다. 자세한 내용은 이미지에서 감지된 PPE 요약 단원을 참조하십시오.

> **Note**
>
> Amazon Rekognition PPE 탐지는 얼굴 인식이나 얼굴 비교를 수행하지 않으므로 탐지된 사람을 식별할 수는 없습니다.

PPE의 유형

DetectProtectiveEquipment는 다음 유형의 PPE를 감지합니다. 이미지에서 다른 유형의 PPE를 감지하려면 Amazon Rekognition Custom Labels를 사용하여 사용자 지정 모델을 학습시키는 것을 권장합니다. 자세한 내용은 Amazon Rekognition Custom Labels를 참조하세요.

얼굴 덮개

`DetectProtectiveEquipment`는 수술용 마스크, N95 마스크, 천으로 만든 마스크 등 일반적인 얼굴 덮개를 감지할 수 있습니다.

손 덮개

`DetectProtectiveEquipment`는 수술용 장갑 및 안전 장갑과 같은 손 덮개를 감지할 수 있습니다.

머리 덮개

`DetectProtectiveEquipment`는 안전모와 헬멧을 감지할 수 있습니다.

해당 API는 이미지에서 머리 덮개, 손 덮개, 또는 얼굴 덮개가 감지되었음을 알려 줍니다. 이 API는 특정 덮개의 유형에 대한 정보를 반환하지 않습니다. 손 덮개 유형의 '수술용 장갑'을 예로 들 수 있습니다.

PPE 탐지 신뢰도

Amazon Rekognition은 이미지에서 PPE, 사람, 그리고 신체 부위의 존재 여부를 예측합니다. API는 Amazon Rekognition이 예측의 정확성에 대해 얼마나 신뢰하는지를 나타내는 점수(50~100)를 제공합니다.

> **Note**
>
> `DetectProtectiveEquipment` 작업을 사용하여 개인의 권리, 개인 정보 보호 또는 서비스 액세스에 영향을 미치는 결정을 내리려는 경우 조치를 취하기 전에 결과를 사람에게 전달하여 검토 및 확인을 받을 것을 권고합니다.

이미지에서 감지된 PPE 요약

선택 사항으로 이미지에서 탐지된 PPE 항목 및 인물에 대한 요약을 요청할 수 있습니다. 필수 보호 장비(얼굴 덮개, 손 덮개, 머리 덮개)의 목록과 최소 신뢰도 임계값(예: 80%)을 지정할 수 있습니다. 응답에는 필수 PPE를 착용한 사람, 필수 PPE를 착용하지 않은 사람, 착용 여부를 알 수 없는 사람의 이미지별 식별자(ID) 통합 요약이 포함됩니다.

이 요약을 통해 얼굴 덮개를 착용하지 않은 사람이 몇 명입니까? 또는 모든 사람이 PPE를 착용하고 있나요?와 같은 질문에 빠르게 답할 수 있습니다. 요약에서 감지된 모든 사람은 고유한 ID를 가집니다. 이 ID를 사용하여 PPE를 착용하지 않은 사람의 경계 상자 위치와 같은 정보를 확인할 수 있습니다.

> **Note**
>
> ID는 이미지별 분석을 기반으로 무작위로 생성되며 이미지 간 또는 동일한 이미지에 대한 여러 분석 결과에서 일관적이지 않습니다.

얼굴 덮개, 머리 덮개, 손 덮개, 또는 원하는 조합을 요약할 수 있습니다. 필수 PPE 유형을 지정하려면 요약 요구 사항 지정 섹션을 참조하세요. 또한 요약에 탐지가 포함되기 위해 충족해야 하는 최소 신뢰도 수준(50~100)을 지정할 수 있습니다.

`DetectProtectiveEquipment`의 요약 응답에 대한 자세한 내용은 DetectProtectiveEquipment 응답 이해 섹션을 참조하세요.

자습서: PPE를 사용하여 이미지를 감지하는 AWS Lambda 함수 생성

Amazon S3 버킷에 있는 이미지에서 개인 보호 장비(PPE)를 감지하는 AWS Lambda 함수를 생성할 수 있습니다. 이 Java V2 자습서를 보려면 AWS 설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

개인용 보호 장비 탐지 API 이해

다음 정보는 DetectProtectiveEquipment API에 대해 설명합니다. 예제 코드는 이미지에서 개인 보호 장비 감지 항목을 참조하세요.

이미지 제공

입력 이미지(JPG 또는 PNG 형식)를 이미지 바이트로 제공하거나 Amazon S3 버킷에 저장된 이미지를 참조할 수 있습니다.

사람의 얼굴이 카메라를 향하고 있는 이미지를 사용하는 것이 좋습니다.

입력 이미지가 0도 방향으로 회전되지 않은 경우, DetectProtectiveEquipment에 제출하기 전에 0도 방향으로 회전시키는 것을 권장합니다. JPG 형식의 이미지에는 교환 이미지 파일 형식(Exif) 메타데이터의 방향 정보가 포함될 수 있습니다. 이 정보를 사용하여 이미지를 회전시키는 코드를 작성할 수 있습니다. 자세한 내용은 Exif 버전 2.32를 참조하세요. PNG 형식 이미지에는 이미지 방향 정보가 포함되어 있지 않습니다.

Amazon S3 버킷에서 이미지를 전달하려면 최소한 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 사용해야 합니다. DetectProtectiveEquipment.를 직접 호출할 수 있는 AmazonRekognitionFullAccess 권한이 있는 사용자를 사용하세요.

다음 예제 입력 JSON에서는 Amazon S3 버킷으로 이미지가 전달됩니다. 자세한 내용은 이미지 작업 단원을 참조하십시오. 이 예제에서는 최소 탐지 신뢰도(MinConfidence)가 80%인 모든 PPE 유형(머리 덮개, 손 덮개, 얼굴 덮개)의 요약을 요청합니다. DetectProtectiveEquipment는 탐지 신뢰도가 50%~100% 사이인 경우에만 예측을 반환하므로 50%~100% 사이의 MinConfidence 값을 지정해야 합니다. 50% 미만의 값을 지정하는 경우 값을 50%로 지정하는 것과 결과는 동일합니다. 자세한 내용은 요약 요구 사항 지정 단원을 참조하십시오.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "worker.jpg"
        }
    },
    "SummarizationAttributes": {
        "MinConfidence": 80,
        "RequiredEquipmentTypes": [
            "FACE_COVER",
```

```
            "HAND_COVER",
            "HEAD_COVER"
        ]
    }
}
```

처리할 이미지 컬렉션이 대규모인 경우, AWS Batch를 사용해 백그라운드에서 일괄적으로 DetectProtectiveEquipment에 대한 직접 호출을 처리하는 방법을 고려하세요.

요약 요구 사항 지정

선택 사항으로 SummarizationAttributes(ProtectiveEquipmentSummarizationAttributes) 입력 파라미터를 사용하여 이미지에서 탐지된 PPE 유형에 대한 요약 정보를 요청할 수 있습니다.

요약할 PPE 유형을 지정하려면 RequiredEquipmentTypes 배열 필드를 사용하세요. 배열에 FACE_COVER, HAND_COVER, HEAD_COVER 중 적어도 하나를 포함하세요.

MinConfidence 필드를 사용하여 최소 탐지 신뢰도(50~100)를 지정합니다. MinConfidence보다 낮은 신뢰도로 탐지된 인물, 신체 부위, 가려진 신체 부위 및 PPE 항목은 요약에 포함되지 않습니다.

DetectProtectiveEquipment 응답에 대한 자세한 내용은 DetectProtectiveEquipment 응답 이해 섹션을 참조하세요.

DetectProtectiveEquipment 응답 이해

DetectProtectiveEquipment는 입력 이미지에서 감지된 인물의 배열을 반환합니다. 각 개인에 대해 탐지된 신체 부위 및 탐지된 PPE 항목에 대한 정보가 반환됩니다. 머리 덮개, 손 덮개, 얼굴 덮개를 착용한 근로자의 아래 이미지에 대한 JSON은 다음과 같습니다.

JSON에서 다음 사항에 유의하세요.

- 탐지된 인물 - Persons는 이미지에서 탐지된 인물의 배열입니다(PPE 미착용 인물 포함). DetectProtectiveEquipment는 이미지에서 감지된 최대 15명의 사람이 착용한 PPE를 감지할 수 있습니다. 배열 내 각 ProtectiveEquipmentPerson 객체에는 인물 ID, 해당 인물의 경계 상자, 감지된 신체 부위, 감지된 PPE 항목이 포함되어 있습니다. ProtectiveEquipmentPerson의 Confidence 값은 Amazon Rekognition이 경계 상자에 사람이 포함되어 있다고 확신하는 정도의 백분율을 나타냅니다.
- 신체 부위 - BodyParts는 사람에게서 감지된 신체 부위(ProtectiveEquipmentBodyPart) 배열입니다(PPE를 착용하고 있지 않은 신체 부위 포함). 각 ProtectiveEquipmentBodyPart에는 탐지된 신체 부위의 이름(Name)이 포함되어 있습니다. DetectProtectEquipment는 얼굴, 머리, 왼손 및 오른손의 신체 부위를 감지할 수 있습니다. ProtectiveEquipmentBodyPart의 Confidence 필드는 신체 부위의 감지 정확도에 대한 Amazon Rekognition의 신뢰도 백분율을 나타냅니다.

- PPE 항목 - ProtectiveEquipmentBody Part 객체의 EquipmentDetections 배열에는 탐지된 PPE 항목 배열이 포함되어 있습니다. 각 EquipmentDetection 객체는 다음 필드를 포함합니다.
  - Type - 감지된 PPE의 유형
  - BoundingBox - 감지된 PPE 주변의 경계 상자
  - Confidence - 경계 상자 안에 탐지된 PPE가 있다고 Amazon Rekognition이 믿는 신뢰도
  - CoversBodyPart - 감지된 PPE가 해당 신체 부위를 덮고 있는지의 여부

  CoversBodyPart 필드 Value는 탐지된 PPE의 해당 신체 부위 착용 여부를 나타내는 부울 값입니다. Confidence 필드는 예측에 대한 신뢰도를 나타냅니다. 탐지된 PPE가 이미지 내에 있지만 실제로 사람이 착용하고 있지는 않은 경우를 필터링하는 데 CoversBodyPart를 사용할 수 있습니다.

  > **Note**
  >
  > CoversBodyPart는 해당 인물이 보호 장비로 적절하게 보호되고 있거나 보호 장비 자체가 제대로 착용되었음을 나타내거나 암시하지 않습니다.

- 요약 정보 - Summary는 SummarizationAttributes 입력 파라미터에 지정된 요약 정보를 포함합니다. 자세한 내용은 요약 요구 사항 지정 단원을 참조하십시오.

  Summary는 다음 정보를 포함하는 ProtectiveEquipmentSummary 유형의 객체입니다.
  - PersonsWithRequiredEquipment - 다음 기준을 각각 충족하는 인물의 ID 배열
    - 해당 인물은 SummarizationAttributes 입력 파라미터에 지정된 PPE를 모두 착용하고 있습니다.
    - 사람(ProtectiveEquipmentPerson), 신체 부위(ProtectiveEquipmentBodyPart), 보호 장비(EquipmentDetection)의 Confidence는 지정된 최소 신뢰도 임계값(MinConfidence)과 같거나 그보다 큽니다.
    - 모든 PPE 항목의 CoversBodyPart 값은 true입니다.
  - PersonsWithoutRequiredEquipment - 다음 기준 중 하나를 충족하는 인물의 ID 배열
    - 사람(ProtectiveEquipmentPerson), 신체 부위(ProtectiveEquipmentBodyPart) 및 가려진 신체 부위(CoversBodyPart)의 Confidence 값이 지정된 최소 신뢰도 임계값(MinConfidence)보다 크지만 해당 인물에서 지정된 PPE(SummarizationAttributes)가 하나 이상 누락되었습니다.
    - Confidence 값이 지정된 최소 신뢰도 임계값(MinConfidence)보다 큰 지정된 PPE(SummarizationAttributes)에서 CoversBodyPart 값이 false입니다. 또한 해당 인물에 지정된 PPE(SummarizationAttributes)가 모두 있으며 사람

(ProtectiveEquipmentPerson), 신체 부위(ProtectiveEquipmentBodyPart) 및 보호 장비(EquipmentDetection)의 Confidence 값이 최소 신뢰도 임계값(MinConfidence)보다 크거나 같습니다.

- PersonsIndeterminate - 사람(ProtectiveEquipmentPerson), 신체 부위(ProtectiveEquipmentBodyPart), 보호 장비(EquipmentDetection)의 Confidence 값 또는 CoversBodyPart 부울 값이 지정된 최소 신뢰도 임계값(MinConfidence)보다 낮은 경우 탐지된 인물의 ID 배열

배열 크기를 사용하여 특정 요약의 개수를 구할 수 있습니다. 예를 들어, PersonsWithRequiredEquipment의 크기는 지정된 유형의 PPE를 착용한 것으로 감지된 사람의 수를 나타냅니다.

인물 ID를 사용하여 경계 상자 위치와 같은 해당 인물에 대한 추가 정보를 확인할 수 있습니다. 인물 ID는 Persons(ProtectiveEquipmentPerson의 배열)로 반환된 ProtectiveEquipmentPerson 객체의 ID 필드에 매핑됩니다. 그러면 대응하는 ProtectiveEquipmentPerson 객체에서 경계 상자와 기타 정보를 가져올 수 있습니다.


```
{
   "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": []
    }
}
```

이미지에서 개인 보호 장비 감지

이미지에서 사람이 착용하고 있는 PPE(개인 보호 장비)를 감지하려면 DetectProtectiveEquipment 비스토리지 API 작업을 사용하세요.

입력 이미지는 AWS SDK나 AWS Command Line Interface (AWS CLI)를 사용하여 이미지 바이트 배열(base64 인코딩된 이미지 바이트) 또는 Amazon S3 객체로 넣을 수 있습니다. 다음 예제에서는 Amazon S3 버킷에 저장된 이미지를 사용합니다. 자세한 내용은 이미지 작업 단원을 참조하십시오.

이미지 내 사람의 PPE를 감지하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. (PPE를 착용하고 있는 한 명 이상의 사람이 포함된) 이미지를 S3 버킷에 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 예제를 사용하여 DetectProtectiveEquipment 작업을 호출합니다. 이미지에 경계 상자를 표시하는 방법에 대한 자세한 내용은 경계 상자 표시 섹션을 참조하세요.

   Java

   이 예제에서는 이미지 내 감지된 사람에게서 감지된 PPE 항목에 대한 정보를 표시합니다.

   amzn-s3-demo-bucket의 값을 이미지 파일이 저장된 Amazon S3 버킷의 이름으로 바꿉니다. photo의 값을 이미지 파일 이름으로 변경합니다.

```
//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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//snippet-start:[rekognition.java2.detect_ppe.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_ppe.import]

/**
```

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

 public static void main(String[] args) {

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

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

     String sourceImage = args[0];
     String bucketName = args[1];
     Region region = Region.US_WEST_2;
     S3Client s3 = S3Client.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

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

     displayGear(s3, rekClient, sourceImage, bucketName) ;
     s3.close();
     rekClient.close();
     System.out.println("This example is done!");
 }
```

```
// snippet-start:[rekognition.java2.detect_ppe.main]
public static void displayGear(S3Client s3,
                               RekognitionClient rekClient,
                               String sourceImage,
                               String bucketName) {

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

    try {
        ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
ProtectiveEquipmentSummarizationAttributes.builder()
            .minConfidence(80F)
            .requiredEquipmentTypesWithStrings("FACE_COVER", "HAND_COVER",
"HEAD_COVER")
            .build();

        SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
        software.amazon.awssdk.services.rekognition.model.Image souImage =
Image.builder()
            .bytes(sourceBytes)
            .build();

        DetectProtectiveEquipmentRequest request =
DetectProtectiveEquipmentRequest.builder()
            .image(souImage)
            .summarizationAttributes(summarizationAttributes)
            .build();

        DetectProtectiveEquipmentResponse result =
rekClient.detectProtectiveEquipment(request);
        List<ProtectiveEquipmentPerson> persons = result.persons();
        for (ProtectiveEquipmentPerson person: persons) {
            System.out.println("ID: " + person.id());
            List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
            if (bodyParts.isEmpty()){
                System.out.println("\tNo body parts detected");
            } else
                for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                    System.out.println("\t" + bodyPart.name() + ". Confidence:
" + bodyPart.confidence().toString());
                    List<EquipmentDetection>
equipmentDetections=bodyPart.equipmentDetections();
```

```
                    if (equipmentDetections.isEmpty()){
                        System.out.println("\t\tNo PPE Detected on " +
bodyPart.name());
                    } else {
                        for (EquipmentDetection item: equipmentDetections) {
                            System.out.println("\t\tItem: " + item.type() + ".
Confidence: " + item.confidence().toString());
                            System.out.println("\t\tCovers body part: "
                                    + item.coversBodyPart().value().toString()
+ ". Confidence: " + item.coversBodyPart().confidence().toString());

                            System.out.println("\t\tBounding Box");
                            BoundingBox box =item.boundingBox();
                            System.out.println("\t\tLeft: "
+box.left().toString());
                            System.out.println("\t\tTop: " +
box.top().toString());
                            System.out.println("\t\tWidth: " +
box.width().toString());
                            System.out.println("\t\tHeight: " +
box.height().toString());
                            System.out.println("\t\tConfidence: " +
item.confidence().toString());
                            System.out.println();
                        }
                    }
                }
        }
        System.out.println("Person ID Summary\n-----------------");

        displaySummary("With required equipment",
result.summary().personsWithRequiredEquipment());
        displaySummary("Without required equipment",
result.summary().personsWithoutRequiredEquipment());
        displaySummary("Indeterminate",
result.summary().personsIndeterminate());

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

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

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

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

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

 static void displaySummary(String summaryType,List<Integer> idList) {
     System.out.print(summaryType + "\n\tIDs  ");
     if (idList.size()==0) {
         System.out.println("None");
     } else {
         int count=0;
         for (Integer id: idList ) {
             if (count++ == idList.size()-1) {
                 System.out.println(id.toString());
             } else {
                 System.out.print(id.toString() + ", ");
             }
         }
     }
     System.out.println();
 }
 // snippet-end:[rekognition.java2.detect_ppe.main]
}
```

AWS CLI

이 AWS CLI 명령은 PPE 요약을 요청하고 detect-protective-equipment CLI 작업에 대한 JSON 출력을 표시합니다.

amzn-s3-demo-bucket을 이미지가 저장된 Amazon S3 버킷의 이름으로 변경합니다. input.jpg를 사용하고자 하는 이미지의 이름으로 변경합니다.

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}" \
  --summarization-attributes
 "MinConfidence=80,RequiredEquipmentTypes=['FACE_COVER','HAND_COVER','HEAD_COVER']"
```

이 AWS CLI 명령은 detect-protective-equipment CLI 작업에 대한 JSON 출력을 표시합니다.

amzn-s3-demo-bucket을 이미지가 저장된 Amazon S3 버킷의 이름으로 변경합니다. input.jpg를 사용하고자 하는 이미지의 이름으로 변경합니다.

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}"
```

Python

이 예제에서는 이미지 내 감지된 사람에게서 감지된 PPE 항목에 대한 정보를 표시합니다.

amzn-s3-demo-bucket의 값을 이미지 파일이 저장된 Amazon S3 버킷의 이름으로 바꿉니다. photo의 값을 이미지 파일 이름으로 변경합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def detect_ppe(photo, bucket):

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

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

 SummarizationAttributes={'MinConfidence': 80,

 'RequiredEquipmentTypes': ['FACE_COVER',

                     'HAND_COVER',

                     'HEAD_COVER']})

   print('Detected PPE for people in image ' + photo)
   print('\nDetected people\n---------------')
   for person in response['Persons']:

       print('Person ID: ' + str(person['Id']))
       print('Body Parts\n----------')
       body_parts = person['BodyParts']
       if len(body_parts) == 0:
           print('No body parts found')
       else:
           for body_part in body_parts:
               print('\t' + body_part['Name'] + '\n\t\tConfidence: ' +
 str(body_part['Confidence']))
               print('\n\t\tDetected PPE\n\t\t------------')
               ppe_items = body_part['EquipmentDetections']
               if len(ppe_items) == 0:
                   print('\t\tNo PPE detected on ' + body_part['Name'])
               else:
                   for ppe_item in ppe_items:
                       print('\t\t' + ppe_item['Type'] + '\n\t\t\tConfidence: '
 + str(ppe_item['Confidence']))
                       print('\t\tCovers body part: ' + str(
                           ppe_item['CoversBodyPart']['Value']) + '\n\t\t
\tConfidence: ' + str(
                           ppe_item['CoversBodyPart']['Confidence']))
                       print('\t\tBounding Box:')
                       print('\t\t\tTop: ' + str(ppe_item['BoundingBox']
['Top']))
                       print('\t\t\tLeft: ' + str(ppe_item['BoundingBox']
['Left']))
                       print('\t\t\tWidth: ' + str(ppe_item['BoundingBox']
['Width']))
```

```
                        print('\t\t\tHeight: ' + str(ppe_item['BoundingBox']
['Height']))
                        print('\t\t\tConfidence: ' +
 str(ppe_item['Confidence']))
            print()
        print()

    print('Person ID Summary\n----------------')
    display_summary('With required equipment', response['Summary']
['PersonsWithRequiredEquipment'])
    display_summary('Without required equipment', response['Summary']
['PersonsWithoutRequiredEquipment'])
    display_summary('Indeterminate', response['Summary']
['PersonsIndeterminate'])

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

# Display summary information for supplied summary.
def display_summary(summary_type, summary):
    print(summary_type + '\n\tIDs: ', end='')
    if (len(summary) == 0):
        print('None')
    else:
        for num, id in enumerate(summary, start=0):
            if num == len(summary) - 1:
                print(id)
            else:
                print(str(id) + ', ', end='')

def main():
    photo = 'photo-name'
    bucket = 'amzn-s3-demo-bucket'
    person_count = detect_ppe(photo, bucket)
    print("Persons detected: " + str(person_count))

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

예제: 얼굴 덮개 주위에 경계 상자 그리기

다음 예제는 인물에서 감지된 얼굴 덮개 주위에 경계 상자를 그리는 방법을 보여줍니다. AWS Lambda 및 Amazon DynamoDB를 사용하는 예제는 AWS 설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

얼굴 덮개를 감지하려면 DetectProtectiveEquipment 비스토리지 API 작업을 사용합니다. 이미지는 로컬 파일 시스템에서 로드됩니다. 입력 이미지를 이미지 바이트 배열(base64 인코딩 이미지 바이트)로 `DetectProtectiveEquipment`에 제공합니다. 자세한 내용은 이미지 작업 단원을 참조하십시오.

이 예제에서는 감지된 얼굴 덮개 주위에 경계 상자를 표시합니다. 얼굴 덮개가 신체 부위를 완전히 덮으면 경계 상자는 녹색입니다. 그렇지 않으면 빨간색 경계 상자가 표시됩니다. 경고 메시지로, 탐지 신뢰도가 지정된 신뢰도 값보다 낮으면 얼굴 덮개 경계 상자 내에 노란색 경계 상자가 표시됩니다. 얼굴 덮개가 감지되지 않으면 인물 주위에 빨간색 경계 상자가 그려집니다.

이미지 출력 결과는 다음과 비슷합니다.

감지된 얼굴 덮개에 경계 상자를 표시하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 DetectProtectiveEquipment 작업을 호출합니다. 이미지에 경계 상자를 표시하는 방법에 대한 자세한 내용은 경계 상자 표시 섹션을 참조하세요.

   Java

   main 함수에서 다음을 변경하세요.

   - photo 값을 로컬 이미지 파일(PNG 또는 JPEG)의 경로 및 파일 이름으로 변경
   - confidence 값을 원하는 신뢰도 수준(50~100)으로 변경

```
//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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
//snippet-end:[rekognition.java2.display_mask.import]
```

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

 DetectProtectiveEquipmentResponse result;
 static BufferedImage image;
 static int scale;
 float confidence;

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

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

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

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

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

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

 // snippet-start:[rekognition.java2.display_mask.main]
public static void displayGear(S3Client s3,
                                    RekognitionClient rekClient,
                                    String sourceImage,
                                    String bucketName) {
    float confidence = 80;
    byte[] data = getObjectBytes(s3, bucketName, sourceImage);
    InputStream is = new ByteArrayInputStream(data);

    try {
        ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
 ProtectiveEquipmentSummarizationAttributes.builder()
            .minConfidence(70F)
            .requiredEquipmentTypesWithStrings("FACE_COVER")
            .build();

        SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
        image = ImageIO.read(sourceBytes.asInputStream());

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

        DetectProtectiveEquipmentRequest request =
 DetectProtectiveEquipmentRequest.builder()
            .image(souImage)
            .summarizationAttributes(summarizationAttributes)
            .build();

        DetectProtectiveEquipmentResponse result =
 rekClient.detectProtectiveEquipment(request);
        JFrame frame = new JFrame("Detect PPE");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        PPEBoundingBoxFrame panel = new PPEBoundingBoxFrame(result, image,
 confidence);
```

```
        panel.setPreferredSize(new Dimension(image.getWidth() / scale,
 image.getHeight() / scale));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    } catch (RekognitionException e) {
        e.printStackTrace();
        System.exit(1);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

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

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

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

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

 public PPEBoundingBoxFrame(DetectProtectiveEquipmentResponse ppeResult,
 BufferedImage bufImage, float requiredConfidence) {
     super();
     scale = 1; // increase to shrink image size.
     result = ppeResult;
     image = bufImage;
     confidence=requiredConfidence;
 }
```

```
// Draws the bounding box around the detected masks.
public void paintComponent(Graphics g) {
    float left = 0;
    float top = 0;
    int height = image.getHeight(this);
    int width = image.getWidth(this);
    int offset=20;

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

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

    // Iterate through detected persons and display bounding boxes.
    List<ProtectiveEquipmentPerson> persons = result.persons();
    for (ProtectiveEquipmentPerson person: persons) {

        List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
        if (!bodyParts.isEmpty()){
            for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                List<EquipmentDetection>
equipmentDetections=bodyPart.equipmentDetections();
                for (EquipmentDetection item: equipmentDetections) {

                    String myType = item.type().toString();
                    if (myType.compareTo("FACE_COVER") ==0) {

                        // Draw green bounding box depending on mask coverage.
                        BoundingBox box =item.boundingBox();
                        left = width * box.left();
                        top = height * box.top();
                        Color maskColor=new Color( 0, 212, 0);

                        if (item.coversBodyPart().equals(false)) {
                            // red bounding box.
                            maskColor=new Color( 255, 0, 0);
                        }
                        g2d.setColor(maskColor);
                        g2d.drawRect(Math.round(left / scale), Math.round(top /
scale),
                                Math.round((width * box.width()) / scale),
Math.round((height * box.height())) / scale);
```

```
                        // Check confidence is > supplied confidence.
                        if (item.coversBodyPart().confidence() < confidence) {
                            // Draw a yellow bounding box inside face mask
 bounding box.
                            maskColor=new Color( 255, 255, 0);
                            g2d.setColor(maskColor);
                            g2d.drawRect(Math.round((left + offset) / scale),
                                    Math.round((top + offset) / scale),
                                    Math.round((width * box.width())- (offset *
 2 ))/ scale,
                                    Math.round((height * box.height()) -
( offset* 2)) / scale);
                        }
                    }
                }
            }
        }
    }
 }
 // snippet-end:[rekognition.java2.display_mask.main]
}
```

Python

`main` 함수에서 다음을 변경하세요.

- `photo` 값을 로컬 이미지 파일(PNG 또는 JPEG)의 경로 및 파일 이름으로 변경
- `confidence` 값을 원하는 신뢰도 수준(50~100)으로 변경
- Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
#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

다음 CLI 예제에서 아래 나열된 인수의 값을 변경합니다.

- photo 값을 로컬 이미지 파일(PNG 또는 JPEG)의 경로 및 파일 이름으로 변경
- confidence 값을 원하는 신뢰도 수준(50~100)으로 변경
- Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
                                aws rekognition detect-protective-equipment
 --image "{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-name"}}" --
profile profile-name \
                                --summarization-attributes
 "{"MinConfidence":MinConfidenceNumber,"RequiredEquipmentTypes":["FACE_COVER"]}"
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
                                aws rekognition detect-protective-equipment  --
image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"image-name
\"}}" \
                                --profile profile-name --summarization-
attributes "{\"MinConfidence\":MinConfidenceNumber,\"RequiredEquipmentTypes\":
[\"FACE_COVER\"]}"
```

유명 인사 인식

Amazon Rekognition을 사용하면 고객은 쉽게 기계 학습을 사용하여 이미지와 동영상에서 수만 명의 유명 인사를 자동으로 인식할 수 있습니다. 유명 인사 인식 API에서 제공하는 메타데이터는 콘텐츠에 태그를 지정하고 검색을 쉽게 만드는 데 필요한 반복적인 수동 작업을 크게 줄여줍니다.

이미지 및 동영상 콘텐츠의 급속한 확산으로 인해 미디어 회사는 자사 미디어 카탈로그를 대규모로 구성, 검색 및 활용하는 데 자주 어려움을 겪습니다. 뉴스 채널과 스포츠 방송사는 현 상황에 대응하고 관련 프로그램을 제작하기 위해 이미지와 동영상을 빠르게 찾아야 할 때가 많습니다. 메타데이터가 충분하지 않으면 이러한 작업이 어려워지지만 Amazon Rekognition을 사용하면 대량의 신규 또는 아카이브 콘텐츠에 자동으로 태그를 지정하여 배우, 운동 선수, 온라인 콘텐츠 크리에이터 등 세계적으로 널리 알려진 광범위한 유명 인사들을 쉽게 검색할 수 있습니다.

Amazon Rekognition 유명 인사 인식은 이미지나 동영상에 알려진 유명 인사가 있을 것으로 예상되는 경우에만 사용하도록 설계되었습니다. 유명 인사가 아닌 얼굴의 인식에 대한 내용은 컬렉션에서 얼굴 검색 단원을 참조하십시오.

> **Note**
>
> 귀하 자신이 유명 인사이고 이 기능에 포함되는 것을 원하지 않으시면 AWS Support에 문의하시거나 <rekognition-celebrity-opt-out@amazon.com>으로 이메일을 보내 주세요.

주제

- 얼굴 검색과 유명 인사 인식 비교
- 이미지 속 유명 인사 인식
- 저장된 동영상 속 유명 인사 인식
- 유명 인사에 대한 정보 얻기

얼굴 검색과 유명 인사 인식 비교

Amazon Rekognition은 유명 인사 인식 기능과 얼굴 인식 기능을 모두 제공합니다. 이들 기능은 사용 사례 및 모범 사례에서 몇 가지 중요한 차이가 있습니다.

유명 인사 인식은 스포츠, 미디어, 정치 및 비즈니스 분야에서 수십만 명이 넘는 유명 인사를 인식할 수 있도록 미리 학습되어 있습니다. 이 기능은 대량의 이미지 또는 비디오를 검색하여 특정 유명 인사가

포함되어 있을 수 있는 소량의 이미지 또는 비디오를 식별하도록 설계되었으며 유명 인사가 아닌 사람들의 얼굴이 일치하는지를 알아보기 위한 것이 아닙니다. 유명 인사 일치의 정확도가 중요한 상황에서는 인간 작업자가 이 기능을 통해 간추려진 콘텐츠를 검토하여 적절한 인간의 판단을 통해 높은 정확도를 확보하는 것이 좋습니다. 유명 인사 인식은 시민적 자유를 침해할 수 있는 방식으로 사용해서는 안 됩니다.

그에 반해, 얼굴 인식은 유명 인사뿐 아니라 모든 개인을 대상으로 신원을 확인하거나 검색하기 위한 자체 얼굴 벡터로 자체 얼굴 컬렉션을 생성할 수 있는 보다 범용적인 기능입니다. 얼굴 인식은 건물 출입 승인, 공공 안전, 소셜 미디어와 같은 애플리케이션에 사용될 수 있습니다. 이러한 사용 사례 모두에서 모범 사례, 적절한 신뢰도(공공 안전 사용 사례의 경우 99%), 일치의 정확도가 중요한 상황에서는 인간 작업자에 의한 검토를 사용하는 것이 좋습니다.

자세한 내용은 컬렉션에서 얼굴 검색 단원을 참조하십시오.

이미지 속 유명 인사 인식

이미지 속 유명 인사를 인식하고, 인식한 유명 인사에 대한 추가 정보를 얻으려면 RecognizeCelebrities 비 스토리지 API 작업을 사용하세요. 예를 들어 정보 수집 타이밍이 중요한 소셜 미디어 또는 뉴스 및 엔터테인먼트 업계에서는 `RecognizeCelebrities` 작업을 사용하여 한 이미지에서 무려 64명의 유명 인사를 식별하고 유명 인사의 웹 페이지가 있다면 그 링크를 반환할 수 있습니다. Amazon Rekognition은 어떤 이미지에서 유명 인사를 감지했는지 기억하지 않습니다. 애플리케이션에서 이 정보를 저장해야 합니다.

`RecognizeCelebrities`에서 반환한 유명 인사에 대한 추가 정보를 저장하지 않은 상태에서 해당 정보를 얻기 위해 이미지를 재분석하는 것을 피하려면 GetCelebrityInfo를 사용하세요. `GetCelebrityInfo`를 직접적으로 호출하려면 Amazon Rekognition이 각 유명 인사에게 할당하는 고유한 식별자가 필요합니다. 식별자는 이미지에서 인식된 각 유명 인사에 대한 `RecognizeCelebrities` 응답의 일부로 반환됩니다.

유명 인사 인식을 위해 처리할 이미지 모음이 많은 경우, AWS Batch를 사용해 백그라운드에서 일괄적으로 `RecognizeCelebrities`에 대한 호출을 처리하는 방법을 고려하십시오. 컬렉션에 새 이미지를 추가할 때 AWS Lambda 함수를 사용하여 이미지가 S3 버킷에 업로드`RecognizeCelebrities`될 때를 호출하여 유명 인사를 인식할 수 있습니다.

RecognizeCelebrities 호출

AWS Command Line Interface () 또는 AWS SDK를 사용하여 입력 이미지를 이미지 바이트 배열(base64 인코딩 이미지 바이트AWS CLI) 또는 Amazon S3 객체로 제공할 수 있습니다. 이 AWS CLI 절차에서는 .jpg 또는 .png 형식의 이미지를 S3 버킷에 업로드합니다. AWS SDK 절차에서는 로컬 파일

시스템에서 로드된 이미지를 사용합니다. 입력 이미지 권장 사항에 대한 자세한 내용은 이미지 작업 단원을 참조하십시오.

이 절차를 실행하려면 에서 한 명 이상의 유명 인사 얼굴이 포함된 이미지 파일이 필요합니다.

이미지에서 유명인 인식

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 RecognizeCelebrities 작업을 호출합니다.

   Java

   이 예제는 이미지에서 감지된 유명 인사에 대한 정보를 표시합니다.

   photo의 값을, 하나 이상의 유명 인사 얼굴을 포함하는 이미지 파일의 경로와 파일 이름으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//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

이 AWS CLI 명령은 `recognize-celebrities` CLI 작업에 대한 JSON 출력을 표시합니다.

`amzn-s3-demo-bucket`을 이미지가 저장된 Amazon S3 버킷의 이름으로 변경합니다. `input.jpg`를 하나 이상의 유명 인사 얼굴을 포함하는 이미지 파일의 이름으로 변경합니다.

`profile_name`의 값을 개발자 프로필 이름으로 바꿉니다.

```
aws rekognition recognize-celebrities \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}"
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
                                    aws rekognition recognize-celebrities --
image \
                                    "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --profile profile-name
```

Python

이 예제는 이미지에서 감지된 유명 인사에 대한 정보를 표시합니다.

photo의 값을, 하나 이상의 유명 인사 얼굴을 포함하는 이미지 파일의 경로와 파일 이름으로 변경합니다.

Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
#Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 이미지에서 감지된 유명 인사에 대한 정보를 표시합니다.

photo의 값을, 하나 이상의 유명 인사 얼굴을 포함하는 이미지 파일의 경로와 파일 이름으로 변경합니다. amzn-s3-demo-bucket의 값을 제공된 이미지 파일이 저장된 S3 버킷의 이름으로 바꿉니다. REGION의 값을 사용자와 연결된 지역 이름으로 변경합니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
// 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

이 예제는 이미지에서 감지된 유명 인사에 대한 정보를 표시합니다.

photo의 값을, 하나 이상의 유명 인사 얼굴(.jpg 또는 .png 형식)을 포함하는 이미지 파일의 경로와 파일 이름으로 변경합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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. 표시되는 유명 인사 ID 중 하나의 값을 기록합니다. 유명 인사에 대한 정보 얻기에서 이 값이 필요할 것입니다.

RecognizeCelebrities 작업 요청

`RecognizeCelebrities`에 대한 입력은 이미지입니다. 이 예에서는 이미지가 이미지 바이트로 전달됩니다. 자세한 내용은 이미지 작업 단원을 참조하십시오.

```
{
    "Image": {
        "Bytes": "/AoSiyvFpm....."
    }
}
```

RecognizeCelebrities 작업 응답

다음은 `RecognizeCelebrities`에 대한 예제 JSON 입력 및 출력입니다.

`RecognizeCelebrities`는 인식된 유명 인사의 배열과 인식되지 않는 얼굴의 배열을 반환합니다. 예제에서 다음 사항에 유의하십시오.

- 인식된 유명 인사 – `Celebrities`는 인식된 유명 인사의 배열입니다. 배열의 각 Celebrity 객체에는 유명 인사 이름과 관련 콘텐츠를 가리키는 URL 목록(예: 유명인의 IMDB 또는 Wikidata 링크)이 포함됩니다. Amazon Rekognition은 애플리케이션이 이미지에서 유명 인사의 얼굴 위치를 확인하는 데 사용할 수 있는 ComparedFace 객체와 유명 인사의 고유 식별자를 반환합니다. 나중에 GetCelebrityInfo API 작업으로 유명 인사 정보를 검색하려면 이 고유한 식별자를 사용합니다.
- 인식되지 않는 얼굴 – `UnrecognizedFaces`는 알려진 유명 인사 누구와도 일치하지 않는 얼굴의 배열입니다. 배열의 각 ComparedFace 객체에는 이미지 속에서 얼굴을 찾는 데 사용할 수 있는 경계 상자가(기타 정보와 함께) 포함되어 있습니다.

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

저장된 동영상 속 유명 인사 인식

Amazon Rekognition Video의 저장된 동영상 속 유명 인사 인식은 비동기 작업입니다. 저장된 동영상에서 유명 인사를 인식하려면 StartCelebrityRecognition을 사용하여 동영상 분석을 시작하세요. Amazon Rekognition Video는 비디오 분석의 완료 상태를 Amazon Simple Notification Service 주제에 게시합니다. 동영상 분석이 성공적으로 완료되면 GetCelebrityRecognition을 직접 호출하여 분석 결과를 가져옵니다. 비디오 분석 시작 및 결과 가져오기에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오.

이 절차는 동영상 분석 요청의 완료 상태를 가져오기 위해 Amazon SQS 대기열을 사용하는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)의 코드를 확장합니다. 이 절차를 실행하려면 한 명 이상의 유명 인사 얼굴이 포함된 비디오 파일이 필요합니다.

Amazon S3 버킷(SDK)에 저장된 동영상에서 유명 인사를 감지하려면

1. Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)을 수행합니다.
2. 1단계에서 만든 클래스 `VideoDetect`에 다음 코드를 추가합니다.

   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);

      }
```

`main` 함수에서 다음 줄을

```
      StartLabelDetection(amzn-s3-demo-bucket, video);

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

다음으로 바꿉니다.

```
        StartCelebrityDetection(amzn-s3-demo-bucket, video);

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

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//snippet-start:[rekognition.java2.recognize_video_celebrity.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.CelebrityRecognitionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.CelebrityRecognition;
import software.amazon.awssdk.services.rekognition.model.CelebrityDetail;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionResponse;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_celebrity.import]

/**
*  To run this code example, ensure that you perform the Prerequisites as stated
 in the Amazon Rekognition Guide:
*  https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
```

```
*
* Also, ensure that set up your development environment, including your
 credentials.
*
* For information, see this documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/

public class RecognizeCelebritiesVideo {

private static String startJobId ="";

public static void main(String[] args) {

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

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

   String bucket = args[0];
   String video = args[1];
   String topicArn = args[2];
   String roleArn = args[3];
   Region region = Region.US_EAST_1;
   RekognitionClient rekClient = RekognitionClient.builder()
       .region(region)
       .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
       .build();

  NotificationChannel channel = NotificationChannel.builder()
```

```
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

  StartCelebrityDetection(rekClient, channel, bucket, video);
  GetCelebrityDetectionResults(rekClient);
  System.out.println("This example is done!");
  rekClient.close();
}

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

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

        StartCelebrityRecognitionRequest recognitionRequest =
 StartCelebrityRecognitionRequest.builder()
            .jobTag("Celebrities")
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartCelebrityRecognitionResponse startCelebrityRecognitionResult =
 rekClient.startCelebrityRecognition(recognitionRequest);
        startJobId = startCelebrityRecognitionResult.jobId();

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

public static void GetCelebrityDetectionResults(RekognitionClient rekClient) {
```

```
  try {
      String paginationToken=null;
      GetCelebrityRecognitionResponse recognitionResponse = null;
      boolean finished = false;
      String status;
      int yy=0 ;

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

          GetCelebrityRecognitionRequest recognitionRequest =
GetCelebrityRecognitionRequest.builder()
              .jobId(startJobId)
              .nextToken(paginationToken)
              .sortBy(CelebrityRecognitionSortBy.TIMESTAMP)
              .maxResults(10)
              .build();

          // Wait until the job succeeds
          while (!finished) {
              recognitionResponse =
rekClient.getCelebrityRecognition(recognitionRequest);
              status = recognitionResponse.jobStatusAsString();

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

          finished = false;

          // Proceed when the job is done - otherwise VideoMetadata is null.
          VideoMetadata videoMetaData=recognitionResponse.videoMetadata();
          System.out.println("Format: " + videoMetaData.format());
          System.out.println("Codec: " + videoMetaData.codec());
          System.out.println("Duration: " + videoMetaData.durationMillis());
          System.out.println("FrameRate: " + videoMetaData.frameRate());
          System.out.println("Job");
```

```
            List<CelebrityRecognition> celebs= recognitionResponse.celebrities();
            for (CelebrityRecognition celeb: celebs) {
                long seconds=celeb.timestamp()/1000;
                System.out.print("Sec: " + seconds + " ");
                CelebrityDetail details=celeb.celebrity();
                System.out.println("Name: " + details.name());
                System.out.println("Id: " + details.id());
                System.out.println();
            }

        } while (recognitionResponse.nextToken() != null);

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

Python

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

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

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

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

        while finished == False:
            response = self.rek.get_celebrity_recognition(JobId=self.startJobId,
```

```
                                                MaxResults=maxResults,
                                                NextToken=paginationToken)

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

            for celebrityRecognition in response['Celebrities']:
                print('Celebrity: ' +
                    str(celebrityRecognition['Celebrity']['Name']))
                print('Timestamp: ' + str(celebrityRecognition['Timestamp']))
                print()

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

main 함수에서 다음 줄을 바꿉니다.

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

다음으로 바꿉니다.

```
    analyzer.StartCelebrityDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetCelebrityDetectionResults()
```

Node.JS

다음 Node.Js 코드 예제에서 amzn-s3-demo-bucket의 값을 동영상이 포함된 S3 버킷의 이름으로 바꾸고 videoName의 값을 해당 동영상 파일의 이름으로 바꾸세요. 또한 roleArn의 값을 IAM 서비스 역할에 연결된 Arn으로 바꿔야 합니다. 마지막으로 region의 값을 귀하의 계정과 연결된 운영 리전의 이름으로 바꾸세요. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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",&TCX5-2025-waiver;
      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

다음 AWS CLI 명령을 실행하여 비디오에서 유명 인사 감지를 시작합니다.

```
aws rekognition start-celebrity-recognition --video "{"S3Object":
{"Bucket":"amzn-s3-demo-bucket","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
--region region-name --profile profile-name
```

다음 값을 업데이트합니다.

- amzn-s3-demo-bucket 및 video-name을 2단계에서 지정한 Amazon S3 버킷 이름과 파일 이름으로 변경합니다.

- region-name을 사용 중인 AWS 리전으로 변경합니다.
- profile-name의 값을 개발자 프로필 이름으로 바꿉니다.
- topic-ARN을 Amazon Rekognition Video 구성의 3단계에서 생성한 Amazon SNS 주제의 ARN으로 변경합니다.
- role-ARN을 Amazon Rekognition Video 구성의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 변경합니다.

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예시를 보려면 다음을 참조하세요.

```
aws rekognition start-celebrity-recognition --video "{\"S3Object\":{\"Bucket\":
\"amzn-s3-demo-bucket\",\"Name\":\"video-name\"}}" \
--notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
--region region-name --profile profile-name
```

진행 중인 코드 예제를 실행한 후 반환된 jobID를 복사하여 다음 GetCelebrityRecognition 명령에 제공하여 결과를 가져오고,job-id-number를 이전에 받은 jobID로 바꾸세요.

```
aws rekognition get-celebrity-recognition --job-id job-id-number --profile
 profile-name
```

> **Note**
>
> Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 이외에 비디오 예제를 이미 실행한 경우, 바꿀 코드가 다를 수 있습니다.

3. 코드를 실행합니다. 비디오에서 인식된 유명 인사에 관한 정보가 표시됩니다.

GetCelebrityRecognition 작업 응답

다음은 JSON 응답의 예입니다. 응답에는 다음이 포함됩니다.

- 인식된 유명 인사 – Celebrities는 동영상 속 유명 인사의 배열과 인식된 시간입니다. 비디오에서 유명 인사가 인식되는 때마다 CelebrityRecognition 객체가 존재합니다. 각 CelebrityRecognition에는 인식된 유명 인사(CelebrityDetail)와 해당 유명 인사가 비디오에서 인식된 시간(Timestamp)에 대한 정보가 포함되어 있습니다. Timestamp는 비디오가 시작된 순간부터 밀리초 단위로 측정됩니다.
- CelebrityDetail – 인식된 유명 인사에 대한 정보가 들어 있습니다. 여기에는 유명 인사 이름(Name), 식별자(ID), 해당 유명 인사의 알려진 성별(KnownGender) 및 관련 콘텐츠를 가리키는 URL 목록(Urls)이 포함됩니다. 또한 Amazon Rekognition Video의 해당 인식에 대한 신뢰도 및 유명 인사의 얼굴인 FaceDetail에 대한 세부 정보도 포함됩니다. 나중에 관련 콘텐츠를 가져와야 하는 경우 getCelebrityInfo와 함께 ID를 사용할 수 있습니다.
- VideoMetadata – 분석된 비디오에 관한 정보입니다.


```
{
    "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
    }
```

```
}
```

유명 인사에 대한 정보 얻기

이 절차에서는 getCelebrityInfo API 작업을 사용하여 유명 인사 정보를 얻습니다. 유명 인사는 이전 RecognizeCelebrities 직접 호출에서 반환된 유명 인사 ID를 사용하여 식별됩니다.

GetCelebrityInfo 호출

이 절차에는 Amazon Rekognition이 알고 있는 유명 인사의 유명 인사 ID가 필요합니다. 이미지 속 유명 인사 인식에서 기록해둔 유명 인사 ID를 사용합니다.

유명 인사 정보를 획득하려면(SDK)

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. `AmazonRekognitionFullAccess` 권한과 `AmazonS3ReadOnlyAccess` 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제를 사용하여 `GetCelebrityInfo` 작업을 호출합니다.

   Java

   이 예제는 유명 인사의 이름과 정보를 표시합니다.

   `id`를 이미지 속 유명 인사 인식에 표시된 유명 인사 ID 중 하나로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

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

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CelebrityInfo {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <id>

                Where:
                   id - The id value of the celebrity. You can use the
 RecognizeCelebrities example to get the ID value.\s
                """;

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

        String id = args[0];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        getCelebrityInfo(rekClient, id);
        rekClient.close();
    }

    /**
     * Retrieves information about a celebrity identified in an image.
     *
     * @param rekClient the Amazon Rekognition client used to make the API call
     * @param id the unique identifier of the celebrity
     * @throws RekognitionException if there is an error retrieving the
 celebrity information
     */
    public static void getCelebrityInfo(RekognitionClient rekClient, String id)
 {
        try {
            GetCelebrityInfoRequest info = GetCelebrityInfoRequest.builder()
                    .id(id)
```

```
                    .build();

            GetCelebrityInfoResponse response =
 rekClient.getCelebrityInfo(info);
            System.out.println("celebrity name: " + response.name());
            System.out.println("Further information (if available):");
            for (String url : response.urls()) {
                System.out.println(url);
            }

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

AWS CLI

이 AWS CLI 명령은 `get-celebrity-info` CLI 작업에 대한 JSON 출력을 표시합니다. `ID`를 이미지 속 유명 인사 인식에 표시된 유명 인사 ID 중 하나로 바꿉니다. `profile-name`의 값을 개발자 프로필 이름으로 바꿉니다.

```
aws rekognition get-celebrity-info --id celebrity-id --profile profile-name
```

Python

이 예제는 유명 인사의 이름과 정보를 표시합니다.

`id`를 이미지 속 유명 인사 인식에 표시된 유명 인사 ID 중 하나로 바꿉니다. Rekognition 세션을 생성하는 라인에서 `profile_name`의 값을 개발자 프로필의 이름으로 대체합니다.

```
# Copyright 2018 Amazon.com, Inc. or its 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

이 예제는 유명 인사의 이름과 정보를 표시합니다.

id를 이미지 속 유명 인사 인식에 표시된 유명 인사 ID 중 하나로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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 작업 요청

다음은 GetCelebrityInfo에 대한 예제 JSON 입력 및 출력입니다.

GetCelebrityInfo에 대한 입력은 해당 유명 인사의 ID입니다.

```
{
    "Id": "nnnnnnn"
}
```

GetCelebrityInfo 작업 응답

GetCelebrityInfo는 요청한 유명 인사의 정보에 대한 링크 배열(Urls)을 반환합니다.

```
{
    "Name": "Celebrity Name",
    "Urls": [
        "www.imdb.com/name/nmnnnnnnn"
    ]
}
```

콘텐츠 조절

Amazon Rekognition을 사용하여 부적절하거나 원치 않거나 불쾌감을 주는 콘텐츠를 탐지할 수 있습니다. 소셜 미디어, 방송 미디어, 광고 및 전자 상거래 현장에서 Rekognition 조절 API를 사용하여 보다 안전한 사용자 경험을 제공하고, 광고주에게 브랜드 안전을 보증하고, 현지 및 글로벌 규정을 준수할 수 있습니다.

오늘날 많은 회사가 제3자 또는 사용자 제작 콘텐츠를 검토할 때 전적으로 인간 모더레이터의 도움을 받기도 하지만 어떤 회사들은 단순히 사용자 불만에 대응하여 불쾌하거나 부적절한 이미지, 광고 또는 동영상을 삭제하기도 합니다. 그러나 인간 모더레이터만으로는 이러한 요구를 충분한 품질 또는 속도로 충족할 수 있도록 규모를 확장할 수 없기 때문에 사용자 경험이 열악해지고 규모 조절에 많은 비용이 들거나 심지어 브랜드 평판이 손상될 수도 있습니다. 이미지 및 동영상 조절에 Rekognition을 사용하면 인간 모더레이터는 기계 학습에 의해 이미 표시된 훨씬 적은 양의 콘텐츠(일반적으로 전체 양의 1~5%)를 검토할 수 있습니다. 이를 통해 보다 가치 있는 활동에 집중하면서도 기존보다 훨씬 적은 비용으로 폭넓은 조절 범위를 확보할 수 있습니다. 작업 인력을 구축하고 인적 검토 작업을 수행하기 위해 Rekognition과 이미 통합된 Amazon Augmented AI를 사용할 수 있습니다.

사용자 지정 조절 기능을 사용하여 딥 러닝 조절 모델의 정확도를 향상시킬 수 있습니다. 사용자 지정 조절을 사용하여 이미지를 업로드하고 주석을 달아 사용자 지정 조절 어댑터를 훈련시키게 됩니다. 그런 다음 훈련된 어댑터를 DetectModerationLabels 작업에 제공하여 이미지에서의 성능을 향상시킬 수 있습니다. 자세한 정보는 사용자 지정 조절을 통한 정확도 향상을 참조하세요.

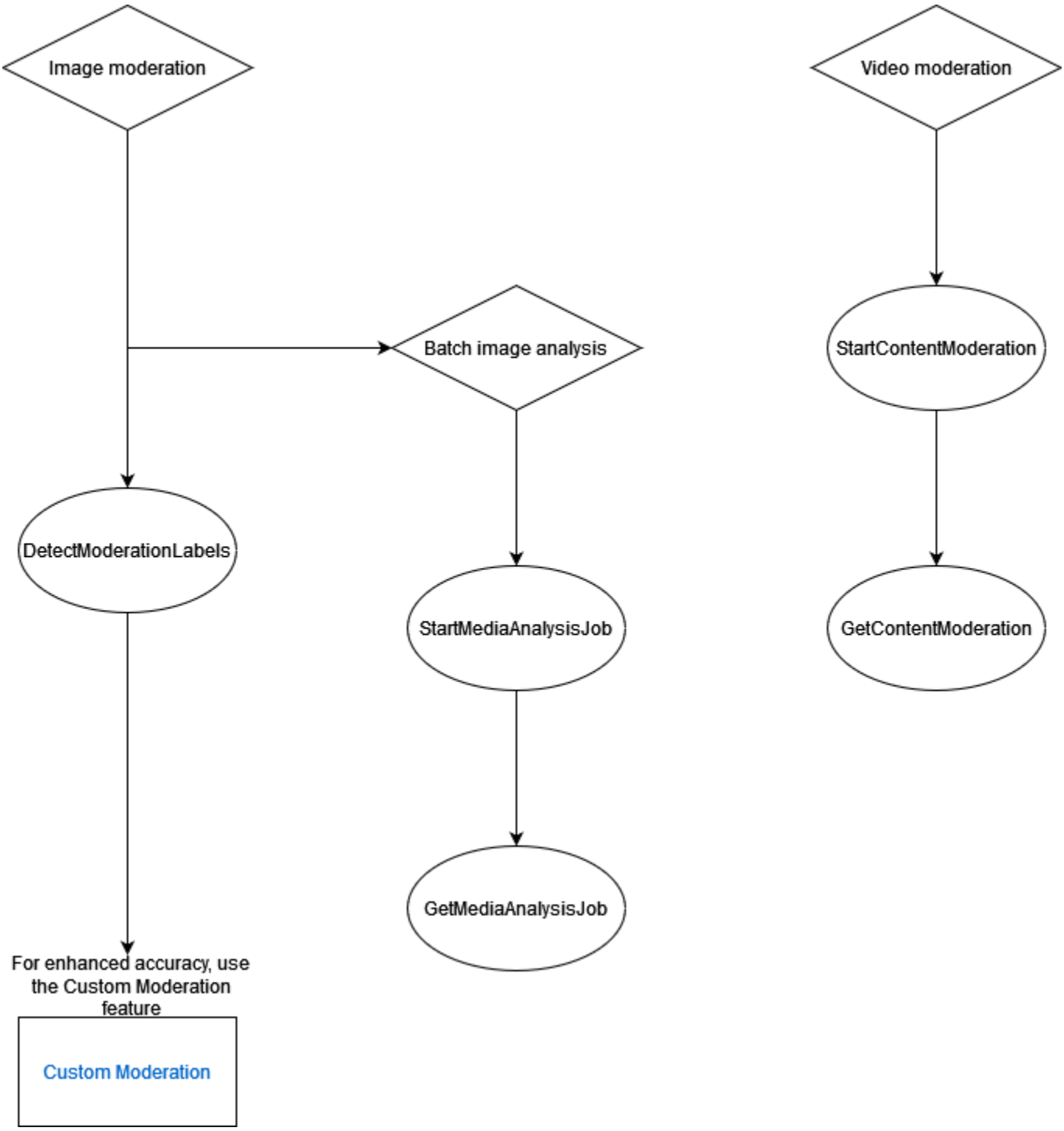
Rekognition 콘텐츠 조절 작업에서 지원하는 레이블

- 조절 레이블 목록을 다운로드하려면 여기를 클릭합니다.

주제

- 이미지 및 비디오 조절 API 사용
- 콘텐츠 조절 버전 7 테스트 및 API 응답 변환
- 부적절한 이미지 감지
- 부적절한 저장된 동영상 탐지
- 사용자 지정 조절을 통한 정확도 향상
- Amazon Augmented AI를 사용한 부적절한 콘텐츠 검토

다음 다이어그램은 콘텐츠 조절의 이미지 또는 비디오 구성 요소 사용 목표에 따른 직접 호출 작업 순서를 보여줍니다.



이미지 및 비디오 조절 API 사용

Amazon Rekognition Image API에서 DetectModerationLabels를 사용하여 부적절하거나 원치 않거나 불쾌감을 주는 콘텐츠를 동기적으로 감지하고 StartMediaAnalysisJob 및 GetMediaAnalysisJob 작업을 사용하여 비동기적으로 감지할 수 있습니다. Amazon Rekognition Video API를 사용하면

StartContentModeration 및 GetContentModeration 작업을 사용하여 이러한 콘텐츠를 비동기적으로 탐지할 수 있습니다.

레이블 카테고리

Amazon Rekognition은 3단계 계층적 분류법을 사용하여 부적절하거나 원치 않거나 불쾌감을 주는 콘텐츠의 카테고리에 레이블을 지정합니다. 분류 수준 1(L1)이 있는 각 레이블에는 여러 분류 수준 2 레이블(L2)이 있으며, 일부 분류 수준 2 레이블에는 분류 수준 3 레이블(L3)이 있을 수 있습니다. 이렇게 하면 콘텐츠의 계층적 분류가 가능합니다.

감지된 각 조절 레이블에 대해 API는 레이블이 속한 수준(1, 2 또는 3)이 포함된 `TaxonomyLevel`도 반환합니다. 예를 들어 이미지는 다음 카테고리에 따라 레이블이 지정될 수 있습니다.

L1: 은밀한 부분의 비노골적 누드 및 키스, L2: 비노골적 누드, L3: 암시적 누드.

> **Note**
>
> L1 또는 L2 카테고리를 사용하여 콘텐츠를 조절하고, L3 카테고리를 사용하여 조절하지 않으려는 특정 개념을 제거하는 것이 좋습니다(즉, 조절 정책에 따라 부적절하거나 원치 않거나 불쾌감을 주는 콘텐츠로 분류하지 않으려는 콘텐츠 감지).

다음 표에는 카테고리 수준과 각 수준에 대해 가능한 레이블 간의 관계가 나와 있습니다. 조절 레이블 목록을 다운로드하려면 여기를 클릭합니다.

최상위 카테고리(L1)	2단계 카테고리(L2)	3단계 카테고리(L3)	정의
Explicit	Explicit Nudity	Exposed Male Genitalia	Human male genitalia , including the penis (whether erect or flaccid), the scrotum, and any discernib le pubic hair. This term is applicable in contexts involving sexual activity or any visual content where male genitals

	are displayed either completely or partially.
Exposed Female Genitalia	External parts of the female reproductive system, encompassing the vulva, vagina, and any observable pubic hair. This term is applicable in scenarios involving sexual activity or any visual content where these aspects of female anatomy are displayed either completely or partially.
Exposed Buttocks or Anus	Human buttocks or anus, including instances where the buttocks are nude or when they are discernible through sheer clothing. The definition specifically applies to situations where the buttocks or anus are directly and completely visible, excluding scenarios where any form of underwear or clothing provides complete or partial coverage.

	Exposed Female Nipple	Human female nipples, including fully visible and partially visible aerola (area surrounding the nipples) and nipples.
Explicit Sexual Activity	N/A	Depiction of actual or simulated sexual acts which encompasses human sexual intercourse, oral sex, as well as male genital stimulation and female genital stimulation by other body parts and objects. The term also includes ejaculation or vaginal fluids on body parts and erotic practices or roleplaying involving bondage, discipline, dominance and submission, and sadomasochism.
Sex Toys	N/A	Objects or devices used for sexual stimulation or pleasure, e.g., dildo, vibrator, butt plug, beats, etc.

Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity	Bare Back	Human posterior part where the majority of the skin is visible from the neck to the end of the spine. This term does not apply when the individua l's back is partially or fully occluded.
		Exposed Male Nipple	Human male nipples, including partially visible nipples.
		Partially Exposed Buttocks	Partially exposed human buttocks. This term includes a partially visible region of the buttocks or butt cheeks due to short clothes, or partially visible top portion of the anal cleft. The term does not apply to cases where the buttocks is fully nude.

		Partially Exposed Female Breast	Partially exposed human female breast where one a portion of the female's breast is visible or uncovered while not revealing the entire breast. This term applies when the region of the inner breast fold is visible or when the lower breast crease is visible with nipple fully covered or occluded.
		Implied Nudity	An individual who is nude, either topless or bottomless, but with intimate parts such as buttocks, nipples, or genitalia covered, occluded, or not fully visible.
	Obstructed Intimate Parts	Obstructed Female Nipple	Visual depiction of a situation in which a female's nipples is covered by opaque clothing or coverings , but their shapes are clearly visible.

		Obstructed Male Genitalia	Visual depiction of a situation in which a male's genitalia or penis is covered by opaque clothing or coverings, but its shape is clearly visible. This term applies when the obstructed genitalia in the image is in close-up.
	Kissing on the Lips	N/A	Depiction of one person's lips making contact with another person's lips.
Swimwear or Underwear	Female Swimwear or Underwear	N/A	Human clothing for female swimwear (e.g., one-piece swimsuits, bikinis, tankinis, etc.) and female underwear (e.g., bras, panties, briefs, lingerie, thongs, etc.)
	Male Swimwear or Underwear	N/A	Human clothing for male swimwear (e.g., swim trunks, boardshorts, swim briefs, etc.) and male underwear (e.g., briefs, boxers, etc.)

Violence	Weapons	N/A	Instruments or devices used to cause harm or damage to living beings, structures, or systems. This includes firearms (e.g., guns, rifles, machine gunes, etc.), sharp weapons (e.g., swords, knives, etc.), explosives and ammunition (e.g., missile, bombs, bullets, etc.).
	Graphic Violence	Weapon Violence	The use of weapons to cause harm, damage, injury, or death to oneself, other individuals, or properties.
		Physical Violence	The act of causing harm to other individuals or property (e.g., hitting, fighting, pulling hair, etc.) or other act of violence involving crowd or multiple individuals.
		Self-Harm	The act of causing harm to oneself, often by cutting body parts such as arms or legs, where cuts are typically visible.

		Blood & Gore	Visual representation of violence on a person, a group of individuals, or animals, involving open wounds, bloodshed, and mutilated body parts.
		Explosions and Blasts	Depiction of a violent and destructive burst of intense flames with thick smoke or dust and smoke erupting from the ground.
Visually Disturbing	Death and Emaciation	Emaciated Bodies	Human bodies that are extremely thin and undernourished with severe physical wasting and depletion of muscle and fat tissue.
		Corpses	Human corpses in the form of mutilated bodies, hanging corpses, or skeletons.

	Crashes	Air Crash	Incidents of air vehicles, such as airplanes, helicopters, or other flying vehicles, resulting in damage, injury, or death. This term applies when parts of the air vehicles are visible.
Drugs & Tobacco	Products	Pills	Small, solid, often round or oval-shaped tables or capsules. This term applies to pills presented as standalones, in a bottle, or a transparent packet and does not apply to a visual depiction of a person taking pills.
	Drugs & Tobacco Paraphernalia & Use	Smoking	The act of inhaling, exhaling, and lighting up burning substances including cigarettes, cigars, e-cigarettes, hookah, or joint.
Alcohol	Alcohol Use	Drinking	The act of drinking alcoholic beverages from bottles or glasses of alcohol or liquor.

	Alcoholic Beverages	N/A	Close up of one or multiple bottles of alcohol or liquor, glasses or mugs with alcohol or liquor, and glasses or mugs with alcohol or liquor held by an individua l. This term does not apply to an individua l drinking from bottles or glasses of alcohol or liquor.
Rude Gestures	Middle Finger	N/A	Visual depiction of a hand gesture with middle finger is extended upward while the other fingers are folded down.
Gambling	N/A	N/A	The act of participa ting in games of chance for a chance to win a prize in casinos, e.g., playing cards, blackjack s, roulette, slot machines at casinos, etc.
Hate Symbols	Nazi Party	N/A	Visual depiction of symbols, flags, or gestures associated with Nazi Party.

White Supremacy	N/A	Visual depiction of symbols or clothings associated with Ku Klux Klan (KKK) and images with confederate flags.
Extremist	N/A	Images containing extremist and terrorist group flags.

L3 카테고리에서 지원되는 레이블이 L2 카테고리의 모든 레이블에 있는 것은 아닙니다. 또한 '제품' 및 '약물 및 담배 관련 용품 및 사용' L2 레이블 아래의 L3 레이블은 전체가 아닙니다. 이러한 L2 레이블은 언급된 L3 레이블 이외의 개념을 다루며, 이러한 경우 API 응답에는 L2 레이블만 반환됩니다.

애플리케이션에 대한 콘텐츠 적합성을 결정합니다. 예를 들어 선정적인 성격의 이미지는 허용하되 누드가 포함된 이미지는 허용하지 않을 수 있습니다. 이미지를 필터링하려면 `DetectModerationLabels`(이미지) 및 `GetContentModeration`(비디오)에서 반환하는 ModerationLabel 레이블 배열을 사용합니다.

콘텐츠 유형

API는 애니메이션 또는 일러스트 콘텐츠 유형을 식별할 수도 있으며, 해당 콘텐츠 유형은 응답의 일부로 반환됩니다.

- 애니메이션 콘텐츠에는 비디오 게임 및 애니메이션(예: 카툰, 만화, 망가, 애니메이션)이 포함됩니다.
- 일러스트 콘텐츠에는 그림, 페인팅, 스케치가 포함됩니다.

신뢰도

`MinConfidence` 입력 파라미터를 지정하여 부적절한 콘텐츠를 감지하기 위해 Amazon Rekognition에서 사용하는 신뢰도 임계값을 설정할 수 있습니다. `MinConfidence`보다 신뢰도가 낮은 것으로 감지된 부적절한 콘텐츠의 레이블은 반환되지 않습니다.

50% 미만인 `MinConfidence`의 값을 지정할 경우 수많은 오탐 결과를 반환할 가능성이 있습니다(즉, 높은 재현률, 낮은 정밀도). 반면, 50%를 초과하는 `MinConfidence`를 지정하면 더 적은 수의 오탐 결

과를 반환할 가능성이 높습니다(즉, 낮은 재현률, 높은 정밀도). MinConfidence 값을 지정하지 않은 경우 Amazon Rekognition에서 신뢰도 50% 이상으로 감지된 부적절한 콘텐츠의 레이블을 반환합니다.

ModerationLabel 배열에는 이전 범주의 레이블 및 인식된 콘텐츠의 정확성에 대한 추정된 신뢰도가 포함되어 있습니다. 식별된 모든 2단계 레이블과 함께 최상위 레이블이 반환됩니다. 예를 들어 Amazon Rekognition에서 신뢰도 점수가 높은 “노골적 나체”를 최상위 레이블로 반환할 수 있습니다. 이는 사용자의 필터링 요건을 충족시키는데 충분할 수 있습니다. 이것으로 필터링에 충분할 수 있지만, 필요하다면 "부분적 누드" 같은 2단계 레이블의 신뢰도 점수를 사용하여 더 세분화된 필터링을 할 수도 있습니다. 예제는 부적절한 이미지 감지 섹션을 참조하세요.

버전 관리

Amazon Rekognition Image와 Amazon Rekognition Video는 모두 부적절한 콘텐츠를 탐지하는 데 사용되는 조절 탐지 모델 버전(ModerationModelVersion)을 반환합니다.

정렬 및 집계

GetContentModeration을 사용하여 결과를 검색할 때 결과를 정렬하고 집계할 수 있습니다.

Sort order - 반환된 레이블의 배열은 시간별로 정렬됩니다. 레이블별로 정렬하려면, GetContentModeration에서 SortBy 입력 파라미터에 NAME를 지정하십시오. 레이블이 비디오에 여러 번 나오는 경우 ModerationLabel 요소의 인스턴스가 여러 개 있습니다.

Label information - ModerationLabels 배열 요소에는 ModerationLabel 객체가 포함되어 있는데 이는 감지된 레이블 이름과 Amazon Rekognition이 그에 대해 가지는 정확성 신뢰도를 포함합니다. 타임스탬프는 ModerationLabel이 감지된 시간으로, 비디오 시작 이후 경과된 밀리초 단위의 시간으로 정의됩니다. 비디오 SEGMENTS별로 집계된 결과의 경우 세그먼트의 시작 시간, 종료 시간, 지속 시간을 각각 정의하는 StartTimestampMillis, EndTimestampMillis, DurationMillis 구조가 반환됩니다.

Aggregation - 결과가 반환될 때 집계되는 방식을 지정합니다. 기본값은 TIMESTAMPS 기준 집계입니다. 일정 기간 동안의 결과를 집계하는 SEGMENTS 집계 기준을 선택할 수도 있습니다. 세그먼트 내에서 감지된 레이블만 반환됩니다.

사용자 지정 조절 어댑터 상태

사용자 지정 조정 어댑터는 TRAINING_IN_PROGRESS, TRAINING_COMPLETED, TRAINING_FAILED, DELETING, DEPRECATED 또는 EXPIRED 상태 중 하나일 수 있습니다. 이러한 어댑터 상태에 대한 자세한 설명은 어댑터 관리를 참조하세요.

> **Note**
>
> Amazon Rekognition은 부적절하거나 불쾌감을 주는 콘텐츠에 대해 공신력이 없으며, 어떤 식으로든 그러한 콘텐츠의 철저한 필터라고 주장하지 않습니다. 추가로 이미지 및 비디오 조절 API는 CSAM과 같은 불법 콘텐츠가 이미지에 포함되어 있는지 여부를 감지하지 않습니다.

콘텐츠 조절 버전 7 테스트 및 API 응답 변환

Rekognition은 콘텐츠 조절 레이블 감지 기능의 이미지 비디오 구성 요소에 대한 기계 학습 모델을 버전 6.1에서 7로 업데이트했습니다. 이 업데이트는 전반적인 정확도를 향상시켰으며 다른 카테고리 수정과 함께 몇 가지 새로운 카테고리를 도입했습니다.

현재 버전 6.1의 비디오 사용자인 경우 버전 7로 원활하게 전환하려면 다음 작업을 수행하는 것이 좋습니다.

1. AWS 프라이빗 SDK(the section called “AWS 콘텐츠 조절 버전 7용 SDK 및 사용 설명서” 참조)를 다운로드하고 사용하여 StartContentModeration API를 직접 호출합니다.
2. API 응답 또는 콘솔에 반환된 레이블 및 신뢰도 점수의 업데이트된 목록을 검토합니다. 필요에 따라 애플리케이션 사후 처리 로직을 조정합니다.
3. 계정은 2024년 5월 13일까지 버전 6.1로 유지됩니다. 2024년 5월 13일 이후 버전 6.1을 사용하려면 2024년 4월 30일까지 AWS Support 팀에 문의하여 연장을 요청하세요. 2024년 6월 10일까지 버전 6.1을 유지하도록 계정을 확장할 수 있습니다. 2024년 4월 30일까지 사용자가 연락하지 않으면 2024년 5월 13일부터 계정이 버전 7.0으로 자동으로 마이그레이션됩니다.

AWS 콘텐츠 조절 버전 7용 SDK 및 사용 설명서

선택한 개발 언어에 해당하는 SDK를 다운로드하고 해당 사용 설명서를 참조하세요.

SDK 링크	설치 / 사용 설명서
Java-1.X	가이드 - Java 1.pdf
Java-2.X	가이드 - Java 2.pdf
JavaScript v2	가이드 - JavaScript v2.pdf

JavaScript v3	가이드 - JavaScript v3.pdf
Python	가이드 - Python & AWS CLI.pdf
Ruby	가이드 - RubyV3.pdf
go_v1	가이드 - GO V1.pdf
go_v2	가이드 - GO V2.pdf
DotNet	가이드 - .NET.pdf
php	가이드 - PHP.pdf

버전 6.1에서 7로 레이블 매핑

콘텐츠 조절 버전 7은 새 레이블 카테고리를 추가하고 이전의 레이블 이름을 수정했습니다. 6.1 레이블을 7 레이블에 매핑하는 방법을 결정할 때는 the section called “레이블 카테고리”의 분류표를 참조하세요.

일부 레이블 매핑 예제는 다음 섹션에서 확인할 수 있습니다. 애플리케이션의 사후 처리 로직을 기반으로 필요한 업데이트를 수행하기 전에 이러한 매핑과 레이블 정의를 검토하는 것이 좋습니다.

L1 매핑 스키마

최상위 카테고리(L1)(예: `Explicit Nudity`, `Suggestive` `Violence` 등)에서만 필터링하는 사후 처리 로직을 사용하는 경우 아래 표를 참조하여 코드를 업데이트하세요.

V6.1 L1	V7 L1
Explicit Nudity	Explicit
Suggestive	Non-Explicit Nudity of Intimate parts and Kissing
	Swimwear or Underwear
Violence	Violence
Visually Disturbing	Visually Disturbing

Rude Gestures	Rude Gestures
Drugs	Drugs & Tobacco
Tobacco	Drugs & Tobacco
Alcohol	Alcohol
Gambling	Gambling
Hate Symbols	Hate Symbols

L2 매핑 스키마

L1 및 L2 카테고리(예: `Explicit Nudity / Nudity, Suggestive / Female Swimwear Or Underwear`, `Violence / Weapon Violence` 등)를 모두 필터링하는 사후 처리 로직을 사용하는 경우 아래 표를 참조하여 코드를 업데이트하세요.

V6.1 L1	V6.1 L2	V7 L1	V7 L2	V7 L3	V7 ContentTypes
Explicit Nudity	Nudity	Explicit	Explicit Nudity	여성 유두 노출	
				둔부 또는 항문 노출	
	Graphic Male Nudity	Explicit	Explicit Nudity	Exposed Male Genitalia	
	Graphic Female Nudity	Explicit	Explicit Nudity	Exposed Female Genitalia	
	Sexual Activity	Explicit	Explicit Sexual Activity		

	Illustrated Explicit Nudity	Explicit	Explicit Nudity		Map to "Animated" and "Illustrated"
	Illustrated Explicit Nudity	Explicit	Explicit Sexual Activity		Map to "Animated" and "Illustrated"
	Adult Toys	Explicit	Sex Toys		
Suggestive	Female Swimwear Or Underwear	Swimwear or Underwear	Female Swimwear or Underwear		
	Male Swimwear Or Underwear	Swimwear or Underwear	Male Swimwear or Underwear		
	Partial Nudity	Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity	Implied Nudity	
	Barechested Male	Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity	Exposed Male Nipple	
	Revealing Clothes	Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity		

		Non-Explicit Nudity of Intimate parts and Kissing	Obstructed Intimate Parts	
	Sexual Situations	Non-Explicit Nudity of Intimate parts and Kissing	Kissing on the Lips	
Violence	Graphic Violence Or Gore	Violence	Graphic Violence	Blood & Gore
	Physical Violence	Violence	Graphic Violence	Physical Violence
	Weapon Violence	Violence	Graphic Violence	Weapon Violence
	Weapons	Violence	Weapons	
	Self Injury	Violence	Graphic Violence	Self-Harm
Visually Disturbing	Emaciated Bodies	Visually Disturbing	Death and Emaciation	Emaciated Bodies
	Corpses	Visually Disturbing	Death and Emaciation	Corpses
	Hanging	Visually Disturbing	Death and Emaciation	Corpses
	Air Crash	Visually Disturbing	Crashes	Air Crash
	Explosions And Blasts	Violence	Graphic Violence	Explosions and Blasts

Rude Gestures	Middle Finger	Rude Gestures	Middle Finger	
Drugs	Drug Products	Drugs & Tobacco	Products	
	Drug Use	Drugs & Tobacco	Drugs & Tobacco Paraphernalia & Use	
	Pills	Drugs & Tobacco	Products	Pills
	Drug Paraphernalia	Drugs & Tobacco	Drugs & Tobacco Paraphernalia & Use	
Tobacco	Tobacco Products	Drugs & Tobacco	Products	
	Smoking	Drugs & Tobacco	Drugs & Tobacco Paraphernalia & Use	Smoking
Alcohol	Drinking	Alcohol	Alcohol Use	Drinking
	Alcoholic Beverages	Alcohol	Alcoholic Beverages	
Gambling	Gambling	Gambling		
Hate Symbols	Nazi Party	Hate Symbols	Nazi Party	
	White Supremacy	Hate Symbols	White Supremacy	
	Extremist	Hate Symbols	Extremist	

부적절한 이미지 감지

DetectModerationLabels 작업을 사용하여 이미지에 부적절하거나 불쾌감을 주는 콘텐츠가 포함되어 있는지 확인할 수 있습니다. Amazon Rekognition의 조절 레이블 목록은 이미지 및 비디오 조절 API 사용을 참조하세요.

이미지에서 부적절한 콘텐츠 감지

이미지는 .jpg 또는 .png 형식이어야 합니다. 입력 이미지를 이미지 바이트 배열(base64 인코딩 이미지 바이트)로 제공하거나 Amazon S3 객체를 지정할 수 있습니다. 이 절차에서는 이미지(.jpg 또는 .png)를 S3 버킷에 업로드합니다.

이러한 절차를 실행하려면 AWS CLI 또는 적절한 AWS SDK가 설치되어 있어야 합니다. 자세한 내용은 Amazon Rekognition 시작 단원을 참조하십시오. 사용할 AWS 계정은 Amazon Rekognition API에 대한 액세스 권한을 가지고 있어야 합니다. 자세한 내용은 Amazon Rekognition에서 정의한 작업을 참조하세요.

이미지에서 조정 레이블(SDK)을 감지하려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. `AmazonRekognitionFullAccess` 권한과 `AmazonS3ReadOnlyAccess` 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. S3 버킷에 이미지를 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 예제를 사용하여 `DetectModerationLabels` 작업을 호출합니다.

   Java

   이 예제는 감지된 부적절한 콘텐츠의 레이블 이름, 신뢰도 수준 및 감지된 조절 레이블의 상위 레이블을 출력합니다.

   `amzn-s3-demo-bucket` 및 `photo` 값을 2단계에서 사용한 S3 버킷 이름과 이미지 파일 이름으로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//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

이 AWS CLI 명령은 detect-moderation-labels CLI 작업에 대한 JSON 출력을 표시합니다.

amzn-s3-demo-bucket 및 input.jpg을 2단계에서 사용한 S3 버킷 이름과 이미지 파일 이름으로 바꿉니다. profile_name의 값을 개발자 프로필 이름으로 바꿉니다. 어댑터를 사용하려면 프로젝트 버전의 ARN을 project-version 파라미터에 제공하세요.

```
aws rekognition detect-moderation-labels --image "{S3Object:{Bucket:<amzn-s3-
demo-bucket>,Name:<image-name>}}" \
--profile profile-name \
--project-version "ARN"
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
aws rekognition detect-moderation-labels --image "{\"S3Object\":{\"Bucket\":
\"amzn-s3-demo-bucket\",\"Name\":\"image-name\"}}" \
--profile profile-name
```

Python

이 예제는 감지된 부적절하거나 불쾌감을 주는 콘텐츠의 레이블 이름, 신뢰도 수준 및 감지된 부적절한 콘텐츠 레이블의 상위 레이블을 출력합니다.

main 함수에서, amzn-s3-demo-bucket 및 photo 값을 2단계에서 사용한 S3 버킷과 이미지의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

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

import boto3

def moderate_image(photo, bucket):

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

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

    print('Detected labels for ' + photo)
    for label in response['ModerationLabels']:
        print (label['Name'] + ' : ' + str(label['Confidence']))
        print (label['ParentName'])
    return len(response['ModerationLabels'])

def main():

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

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

.NET

이 예제는 감지된 부적절하거나 불쾌감을 주는 콘텐츠의 레이블 이름, 신뢰도 수준 및 감지된 조절 레이블의 상위 레이블을 출력합니다.

amzn-s3-demo-bucket 및 photo 값을 2단계에서 사용한 S3 버킷 이름과 이미지 파일 이름으로 바꿉니다.

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

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

public class DetectModerationLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectModerationLabelsRequest detectModerationLabelsRequest = new
 DetectModerationLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MinConfidence = 60F
        };

        try
        {
```

```
            DetectModerationLabelsResponse detectModerationLabelsResponse =
 rekognitionClient.DetectModerationLabels(detectModerationLabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (ModerationLabel label in
 detectModerationLabelsResponse.ModerationLabels)
                Console.WriteLine("Label: {0}\n Confidence: {1}\n Parent: {2}",
                    label.Name, label.Confidence, label.ParentName);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

DetectModerationLabels 작업 요청

`DetectModerationLabels`에 대한 입력은 이미지입니다. 이 예제 JSON 입력에서는 Amazon S3 버킷에서 소스 이미지를 불러옵니다. `MinConfidence`는 Amazon Rekognition Image가 응답에 반환하기 위해 충족해야 할 감지된 레이블의 최소 정확성 신뢰도 수준입니다.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "input.jpg"
        }
    },
    "MinConfidence": 60
}
```

DetectModerationLabels 작업 응답

`DetectModerationLabels`가 S3 버킷의 입력 이미지를 감지하거나, 이미지를 이미지 바이트로 직접 제공할 수 있습니다. 다음 예제는 `DetectModerationLabels` 호출로부터의 응답입니다.

다음 JSON 응답 예제에서 다음에 유의하십시오.

- 부적절한 이미지 감지 정보 - 이 예제에서는 이미지에서 발견된 부적절하거나 불쾌감을 주는 콘텐츠의 레이블 목록을 보여줍니다. 이 목록에는 이미지에서 감지되는 최상위 레이블과 각각의 2수준 레이블이 포함됩니다.

레이블 – 각 레이블에는 이름, 해당 레이블이 정확한지에 대한 Amazon Rekognition의 신뢰도 추정, 상위 레이블의 이름이 있습니다. 최상위 레이블의 상위 이름은 ""입니다.

레이블 신뢰도 – 각 레이블에는 레이블이 올바른지에 대해 Amazon Rekognition이 가지고 있는 백분율 신뢰도를 나타내는 0에서 100 사이의 신뢰도 값이 있습니다. API 작업 요청에서 응답으로 반환될 레이블에 필요한 신뢰도 수준을 지정합니다.


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

부적절한 저장된 동영상 탐지

Amazon Rekognition Video의 저장된 비디오 속 부적절하거나 불쾌감을 주는 콘텐츠 탐지는 비동기 작업입니다. 부적절하거나 불쾌감을 주는 콘텐츠 탐지를 시작하려면 StartContentModeration을 직접 호출하세요. Amazon Rekognition Video는 비디오 분석의 완료 상태를 Amazon Simple Notification Service 주제에 게시합니다. 비디오 분석이 성공적으로 완료되면, GetContentModeration 직접 호출을 통해 분석 결과를 가져옵니다. 비디오 분석 시작 및 결과 가져오기에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오. Amazon Rekognition의 조절 레이블 목록은 이미지 및 비디오 조절 API 사용을 참조하세요.

이 절차는 동영상 분석 요청의 완료 상태를 가져오기 위해 Amazon Simple Queue Service 대기열을 사용하는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)의 코드를 확장합니다.

Amazon S3 버킷(SDK)에 저장된 비디오에서 부적절하거나 불쾌감을 주는 콘텐츠를 탐지하려면

1. Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)을 수행합니다.
2. 1단계에서 만든 클래스 `VideoDetect`에 다음 코드를 추가합니다.

   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);

        }
```

`main` 함수에서 다음 줄을 바꿉니다.

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

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

다음으로 바꿉니다.

```
        StartUnsafeContentDetection(amzn-s3-demo-bucket, video);

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

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
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
```

main 함수에서 다음 줄을 바꿉니다.

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

다음으로 바꿉니다.

```
    analyzer.StartUnsafeContent()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetUnsafeContentResults()
```

> **Note**
>
> Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 이외에 비디오 예제를 이미 실행한 경우, 바꿀 코드가 다를 수 있습니다.

3. 코드를 실행합니다. 비디오에서 감지된 부적절한 콘텐츠 레이블 목록이 표시됩니다.

GetContentModeration 작업 응답

`GetContentModeration`의 응답은 ContentModerationDetection 객체로 구성된 `ModerationLabels` 배열입니다. 부적절한 콘텐츠 레이블이 감지될 때마다 배열에 요소가 포함됩니다. `ContentModerationDetectionObject` 객체 내에 있는 ModerationLabel에는 부적절하거나 불쾌감을 주는 콘텐츠가 감지된 항목에 대한 정보가 포함되어 있습니다. `Timestamp`는 비디오 시작부터 레이블이 감지된 때까지 지난 시간(밀리초 단위)입니다. 레이블은 부적절한 콘텐츠 이미지 분석을 통해 감지된 레이블과 동일한 방식의 계층적 구조로 구성됩니다. 자세한 내용은 콘텐츠 조절 단원을 참조하십시오.

다음은 `NAME` 기준으로 정렬하고 `TIMESTAMPS` 기준으로 집계한 `GetContentModeration`로부터의 예제 응답입니다.

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
```

```
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
        {
            "Timestamp": 36000,
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
                "Name": "Alcohol",
                "ParentName": "",
                "TaxonomyLevel": 1
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        },
        {
            "Timestamp": 36000,
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
                "Name": "Alcoholic Beverages",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video-name.mp4"
        }
    },
```

```
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "TIMESTAMPS"
    }
}
```

다음은 NAME 기준으로 정렬하고 SEGMENTS 기준으로 집계한 GetContentModeration로부터의 예제 응답입니다.

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
        {
            "Timestamp": 0,
            "ModerationLabel": {
                "Confidence": 0.0003000000142492354,
                "Name": "Alcohol Use",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 29520,
            "DurationMillis": 29520,
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Illustrated"
                },
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
```

```
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video-name.mp4"
        }
    },
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "SEGMENTS"
    }
}
```

사용자 지정 조절을 통한 정확도 향상

Amazon Rekognition의 DetectModerationLabels API를 사용하여 부적절하거나 원치 않거나 불쾌감을 주는 콘텐츠를 탐지할 수 있습니다. Rekognition 사용자 지정 조절 기능을 사용하면 어댑터를 사용하여 DetectModerationLabels의 정확도를 높일 수 있습니다. 어댑터는 기존 Rekognition 딥 러닝 모델에 추가할 수 있는 모듈식 구성 요소로, 훈련 대상 작업에 맞게 기능을 확장합니다. 어댑터를 만들어 DetectModerationLabels 작업에 제공하면 특정 사용 사례와 관련된 콘텐츠 조절 작업의 정확도를 높일 수 있습니다.

Rekognition의 콘텐츠 조절 모델을 특정 조절 레이블에 맞게 사용자 지정할 때는 프로젝트를 만들고 제공하는 이미지 세트를 기반으로 어댑터를 훈련시켜야 합니다. 그런 다음 어댑터의 성능을 반복적으로 확인하고 어댑터를 원하는 정확도 수준이 되도록 다시 훈련시킬 수 있습니다. 프로젝트는 다양한 버전의 어댑터를 저장하는 데 사용됩니다.

Rekognition 콘솔을 사용하여 프로젝트 및 어댑터를 생성할 수 있습니다. 또는 AWS SDK 및 연결된 APIs를 사용하여 프로젝트를 생성하고, 어댑터를 훈련하고, 어댑터를 관리할 수 있습니다.

어댑터 생성 및 사용

어댑터는 기존 Rekognition 딥 러닝 모델에 추가할 수 있는 모듈식 구성 요소로, 훈련 대상 작업에 맞게 기능을 확장합니다. 어댑터로 딥 러닝 모델을 훈련시키면 특정 사용 사례와 관련된 이미지 분석 작업의 정확도를 높일 수 있습니다.

어댑터를 만들고 사용하려면 Rekognition에 훈련 및 테스트 데이터를 제공해야 합니다. 다음 두 가지 중 하나의 방법으로 이 작업을 수행할 수 있습니다.

- 대량 분석 및 검증 - Rekognition에서 분석하고 레이블을 할당할 이미지를 대량으로 분석하여 훈련 데이터 세트를 생성할 수 있습니다. 그런 다음 이미지에 대해 생성된 주석을 검토하고 예측을 확인하거나 수정할 수 있습니다. 이미지 대량 분석 작동 방식에 대한 자세한 내용은 대량 분석을 참조하세요.
- 수동 주석 - 이 접근 방식을 사용하면 이미지를 업로드하고 주석을 달아 훈련 데이터를 만들 수 있습니다. 이미지를 업로드하고 주석을 달거나 자동 분할하여 테스트 데이터를 생성합니다.

자세히 알아보려면 다음 주제 중 하나를 선택하세요.

주제

- 대량 분석 및 검증
- 수동 주석

대량 분석 및 검증

이 방법을 사용하면 훈련 데이터로 사용할 이미지를 대량으로 업로드한 다음 이미지에 레이블을 자동으로 할당하는 Rekognition을 사용하여 이미지에 대한 예측을 가져옵니다. 이러한 예측을 어댑터의 시작점으로 사용할 수 있습니다. 예측의 정확성을 확인한 다음 검증된 예측을 기반으로 어댑터를 훈련시킬 수 있습니다. 콘솔을 사용하여이 작업을 수행할 수 있습니다 AWS .

대량 분석 및 사용자 지정 조절

대량 분석을 위한 이미지 업로드

어댑터용 훈련 데이터 세트를 생성하려면 Rekognition에서 레이블을 예측할 이미지를 대량으로 업로드하세요. 최상의 결과를 얻으려면 최대 한도인 10,000개까지 훈련용 이미지를 최대한 많이 제공하고 이미지가 여러분의 사용 사례의 모든 면을 반영하는지 확인하세요.

AWS 콘솔을 사용하는 경우 컴퓨터에서 직접 이미지를 업로드하거나 이미지를 저장하는 Amazon Simple Storage Service 버킷을 제공할 수 있습니다. 하지만 Rekognition API를 SDK와 함께 사용하는 경우 Amazon Simple Storage Service 버킷에 저장된 이미지를 참조하는 매니페스트 파일을 제공해야 합니다. 자세한 내용은 대량 분석을 참조하세요.

예측 검토

Rekognition 콘솔에 이미지 업로드를 완료하면 Rekognition에서 이미지에 대한 레이블을 생성합니다. 그런 다음 예측을 true positive, false positive, true negative, false negative의 카테고리 중 하나로 검증할 수 있습니다. 예측을 검증한 후 이 피드백을 기반으로 어댑터를 훈련시킬 수 있습니다.

어댑터 훈련

대량 분석에서 반환된 예측의 검증을 마치면 어댑터 훈련 프로세스를 시작할 수 있습니다.

AdapterId 가져오기

어댑터 훈련이 완료되면 Rekognition의 이미지 분석 API와 함께 사용할 어댑터의 고유 ID를 얻을 수 있습니다.

API 작업 직접 호출

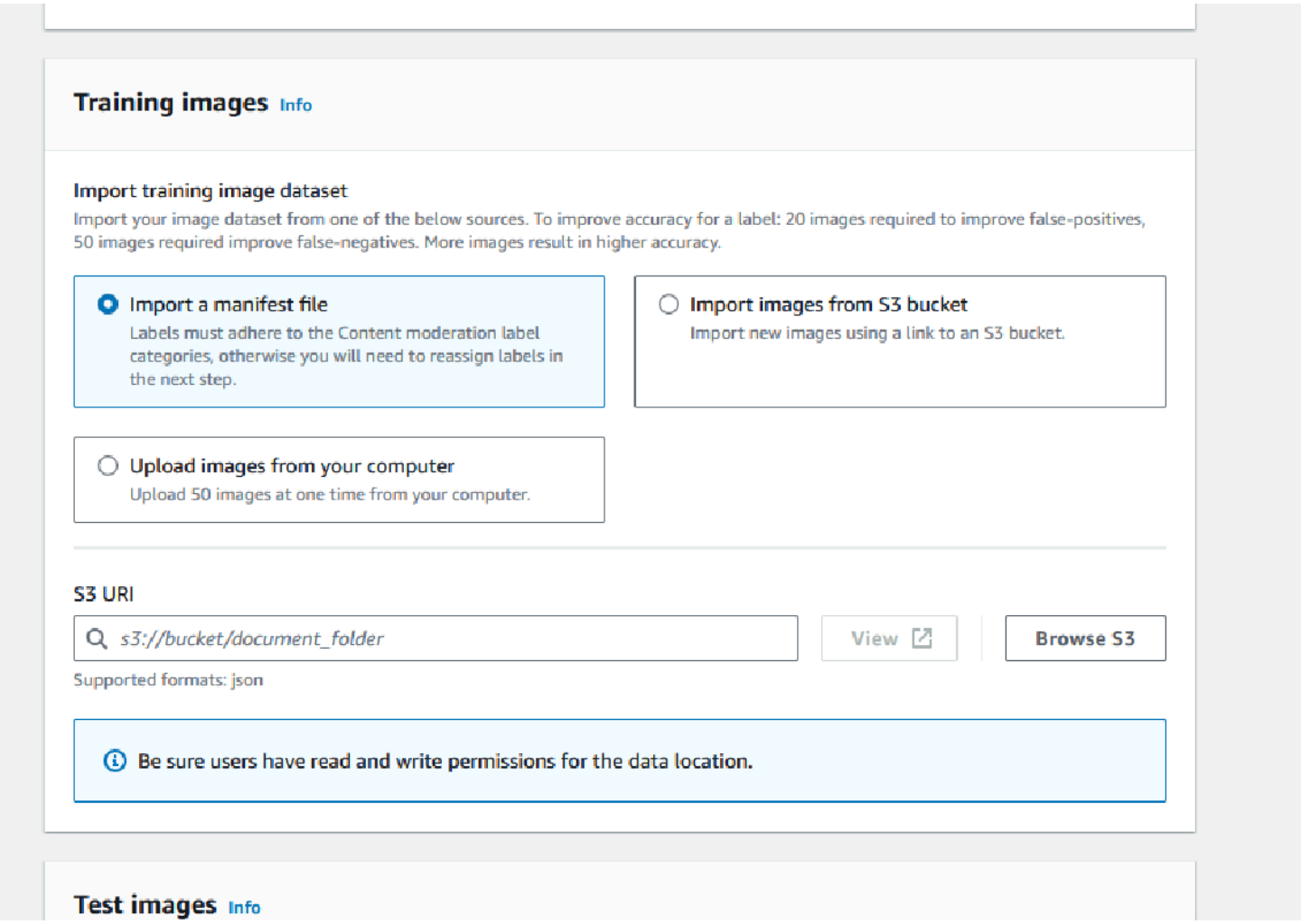
사용자 지정 어댑터를 적용하려면 어댑터를 지원하는 이미지 분석 API 중 하나를 직접 호출할 때 해당 ID를 제공하세요. 이렇게 하면 이미지에 대한 예측의 정확도가 향상됩니다.

수동 주석

이 접근 방식에서는 수동으로 이미지를 업로드하고 주석을 달아 훈련 데이터를 만듭니다. 테스트 이미지를 업로드하고 주석을 달거나 Rekognition이 자동으로 훈련 데이터의 일부를 테스트 이미지로 사용하도록 자동 분할하여 테스트 데이터를 생성합니다.

이미지 업로드 및 주석 달기

어댑터를 훈련시키려면 사용 사례를 반영하는 샘플 이미지 세트를 업로드해야 합니다. 최상의 결과를 얻으려면 최대 한도인 10,000개까지 훈련용 이미지를 최대한 많이 제공하고 이미지가 여러분의 사용 사례의 모든 면을 반영하는지 확인하세요.



AWS 콘솔을 사용할 때 컴퓨터에서 직접 이미지를 업로드하거나, 매니페스트 파일을 제공하거나, 이미지를 저장하는 Amazon S3 버킷을 제공할 수 있습니다.

하지만 Rekognition API를 SDK와 함께 사용하는 경우 Amazon S3 버킷에 저장된 이미지를 참조하는 매니페스트 파일을 제공해야 합니다.

Rekognition 콘솔의 주석 인터페이스를 사용하여 이미지에 주석을 달 수 있습니다. 이미지에 레이블로 태그를 지정해 주석을 달면 훈련에 필요한 "실측 정보"가 확립됩니다. 또한 어댑터를 훈련시키기 전에 반드시 훈련 및 테스트 세트를 지정하거나 자동 분할 기능을 사용해야 합니다. 데이터 세트를 지정하고 이미지에 주석 달기를 완료했다면 테스트 세트의 주석이 달린 이미지를 기반으로 어댑터를 생성할 수 있습니다. 그 후 어댑터의 성능을 평가할 수 있습니다.

테스트 세트 생성

주석이 달린 테스트 세트를 제공하거나 자동 분할 기능을 사용해야 합니다. 훈련 세트는 어댑터를 실제로 훈련시키는 데 사용됩니다. 어댑터는 주석이 달린 이미지에 포함된 패턴을 학습합니다. 테스트 세트는 어댑터를 최종 완성하기 전에 모델의 성능을 평가하는 데 사용됩니다.

어댑터 훈련

훈련 데이터에 주석을 달거나 매니페스트 파일을 제공한 후에는 어댑터에 대한 훈련 프로세스를 시작할 수 있습니다.

어댑터 ID 가져오기

어댑터 훈련이 완료되면 Rekognition의 이미지 분석 API와 함께 사용할 어댑터의 고유 ID를 얻을 수 있습니다.

API 작업 직접 호출

사용자 지정 어댑터를 적용하려면 어댑터를 지원하는 이미지 분석 API 중 하나를 직접 호출할 때 해당 ID를 제공하세요. 이렇게 하면 이미지에 대한 예측의 정확도가 향상됩니다.

데이터 세트 준비

어댑터를 만들려면 Rekognition에 교육 데이터 세트와 테스트 데이터 세트, 두 개의 데이터 세트를 제공해야 합니다. 각 데이터 세트는 이미지와 주석 또는 레이블이라는 두 가지 요소로 구성됩니다. 다음 섹션에서는 레이블과 이미지의 용도와 이들을 결합하여 데이터 세트를 생성하는 방법을 설명합니다.

이미지

이미지의 대표적인 샘플을 기반으로 어댑터를 훈련시켜야 합니다. 훈련용 이미지를 선택할 때는 어댑터로 얻고자 하는 각 레이블의 예상 응답을 보여주는 이미지를 최소한 몇 개 포함시키세요.

훈련 데이터 세트를 만들려면 다음 두 이미지 유형 중 하나를 제공해야 합니다.

- False Positive 예측이 포함된 이미지. 기본 모델이 이미지 안에 술이 있다고 예측했지만 실제로는 그렇지 않은 경우를 예로 들 수 있습니다.
- False Negative 예측이 포함된 이미지. 기본 모델이 이미지 안에 술이 없다고 예측했지만 실제로는 술이 있는 경우를 예로 들 수 있습니다.

균형 잡힌 데이터 세트를 만들려면 다음 두 이미지 유형 중 하나를 제공하는 것이 좋습니다.

- True Positive 예측이 포함된 이미지. 기본 모델이 이미지 안에 술이 있다고 올바르게 예측한 경우를 예로 들 수 있습니다. False Positive 이미지를 제공하는 경우 이러한 이미지도 제공하는 것이 좋습니다.
- True Negative 예측이 포함된 이미지. 기본 모델이 이미지 안에 술이 없다고 올바르게 예측한 경우를 예로 들 수 있습니다. False Negative 이미지를 제공하는 경우 이러한 이미지도 제공하는 것이 좋습니다.

레이블

레이블은 객체, 이벤트, 개념 또는 활동을 가리킵니다. 콘텐츠 조절에서 레이블은 부적절하거나, 원치 않거나, 불쾌감을 주는 콘텐츠의 인스턴스입니다.

Rekognition의 기본 모델을 훈련시켜 어댑터를 만드는 상황에서는 이미지에 레이블이 할당되는 것을 주석이라고 합니다. Rekognition 콘솔로 어댑터를 훈련시키는 경우 콘솔을 사용하여 레이블을 선택한 다음 레이블에 해당하는 이미지에 태그를 지정하여 이미지에 주석을 추가합니다. 이 프로세스를 통해 모델은 할당된 레이블을 기반으로 이미지의 요소를 식별하는 법을 배우게 됩니다. 이 연결 프로세스는 어댑터가 생성되고 나면 모델이 가장 관련성이 높은 콘텐츠에 집중할 수 있도록 하므로 이미지 분석의 정확도가 향상됩니다.

또는 이미지에 대한 정보와 함께 제공되는 주석이 포함된 매니페스트 파일을 제공할 수도 있습니다.

훈련 데이터 세트 및 테스트 데이터 세트

훈련 데이터 세트는 모델을 미세 조절하고 사용자 지정 어댑터를 만들기 위한 기초가 됩니다. 모델이 학습할 수 있도록 주석이 달린 훈련 데이터 세트를 제공해야 합니다. 모델은 이 데이터 세트를 기반으로 학습하여 사용자가 제공하는 이미지 유형에 대한 성능을 개선합니다.

정확도를 높이려면 이미지에 주석을 달거나 레이블을 지정하여 훈련 데이터 세트를 만들어야 합니다. 다음 두 가지 방법으로 이 작업을 수행할 수 있습니다.

- 수동 레이블 할당 - 데이터 세트에 포함할 이미지를 업로드한 후 이러한 이미지에 수동으로 레이블을 할당하여 Rekognition 콘솔을 사용한 훈련 데이터 세트를 생성할 수 있습니다.
- 매니페스트 파일 - 매니페스트 파일을 사용하여 어댑터를 훈련할 수 있습니다. 매니페스트 파일에는 훈련 이미지 및 테스트 이미지의 실측 정보 주석과 훈련 이미지의 위치에 대한 정보가 들어 있습니다. Rekognition APIs를 사용하여 어댑터를 훈련하거나 AWS 콘솔을 사용할 때 매니페스트 파일을 제공할 수 있습니다.

테스트 데이터 세트는 훈련 후 어댑터의 성능을 평가하는 데 사용됩니다. 신뢰성 있는 평가를 보장하기 위해 모델이 본 적 없는 원본 훈련 데이터 세트의 일부를 사용하여 테스트 데이터 세트를 만듭니다. 이 프로세스를 통해 새 데이터로 어댑터의 성능을 평가하여 정확한 측정치 및 지표를 생성할 수 있습니다. 최적 정확도 개선에 대해서는 어댑터 훈련 모범 사례 섹션을 참조하세요.

AWS CLI 및 SDKs를 사용하여 어댑터 관리

Rekognition을 사용하면 사전 훈련된 컴퓨터 비전 모델을 활용하는 여러 기능을 사용할 수 있습니다. 이러한 모델을 사용하면 레이블 감지 및 콘텐츠 조절과 같은 작업을 수행할 수 있습니다. 어댑터를 사용하여 이러한 특정 모델을 사용자 정의할 수도 있습니다.

Rekognition의 프로젝트 생성 및 프로젝트 관리 API(예: CreateProject 및 CreateProjectVersion)를 사용하여 어댑터를 만들고 교육할 수 있습니다. 다음 페이지에서는 AWS 콘솔, 선택한 AWS SDK 또는 AWS CLI를 사용하여 API 작업을 사용하여 어댑터를 생성, 훈련 및 관리하는 방법을 설명합니다.

어댑터를 훈련시킨 후에는 지원되는 기능을 사용하여 추론을 실행할 때 어댑터를 사용할 수 있습니다. 어댑터는 현재 콘텐츠 조절 기능을 사용할 때 지원됩니다.

AWS SDK를 사용하여 어댑터를 훈련할 때는 매니페스트 파일 형식으로 실제 레이블(이미지 주석)을 제공해야 합니다. 또는 Rekognition 콘솔을 사용하여 어댑터를 만들고 훈련할 수 있습니다.

> **Note**
>
> 어댑터는 복사할 수 없습니다. Rekognition Custom Labels 프로젝트 버전만 복사할 수 있습니다.

주제

- 어댑터 상태
- 프로젝트 생성
- 프로젝트 설명
- 프로젝트 삭제
- 프로젝트 버전 생성
- 프로젝트 버전 설명
- 프로젝트 버전 삭제

어댑터 상태

사용자 지정 조절 어댑터(프로젝트 버전)는 다음 상태 중 하나일 수 있습니다.

- TRAINING_IN_PROGRESS - 어댑터는 사용자가 훈련 문서로 제공한 파일에 대해 훈련 중입니다.
- TRAINING_COMPLETED - 어댑터가 훈련을 성공적으로 완료했으며 성능을 검토할 준비가 되었습니다.
- TRAINING_FAILED - 어댑터가 어떤 이유로 훈련을 완료하지 못했습니다. 장애 원인에 대한 자세한 내용은 출력 매니페스트 파일 및 출력 매니페스트 요약을 검토하세요.
- DELETING - 어댑터를 삭제 중입니다.
- DEPRECATED - 어댑터가 이전 버전의 콘텐츠 조절 기본 모델에 대해 훈련되었습니다. 유예 기간 내에 있으며 새 기본 모델 버전 출시 후 60~90일 이내에 만료됩니다. 유예 기간 동안 어댑터를 사용하여 DetectModerationLabels 또는 StartMediaAnalysisJob API 작업을 추론할 수 있습니다. 어댑터의 만료 날짜는 사용자 지정 조절 콘솔을 참조하세요.
- EXPIRED - 어댑터가 이전 버전의 콘텐츠 조절 기본 모델에 대해 훈련되었으며, 더 이상 DetectModerationLabels 또는 StartMediaAnalysisJob API 작업으로 사용자 지정 결과를 얻는 데 사용할 수 없습니다. 추론 요청에 만료된 어댑터가 지정된 경우 해당 어댑터는 무시되고 대신 최신 버전의 사용자 지정 조절 기본 모델에서 응답이 반환됩니다.

프로젝트 생성

CreateProject 작업을 통해 Rekognition의 레이블 감지 작업을 위한 어댑터를 넣을 프로젝트를 만들 수 있습니다. 프로젝트는 리소스 그룹이며 DetectModerationLabels와 같은 레이블 탐지 작업의 경우 프로젝트를 통해 기본 Rekognition 모델을 사용자 지정하는 데 사용할 수 있는 어댑터를 저장할 수 있습니다. CreateProject를 간접 호출할 때는 만들려는 프로젝트의 이름을 ProjectName 인수에 입력합니다.

AWS 콘솔을 사용하여 프로젝트를 생성하려면:

- Rekognition 콘솔에 로그인합니다.
- 사용자 지정 조절을 클릭합니다.
- 프로젝트 생성을 선택합니다.
- 새 프로젝트 생성 또는 기존 프로젝트에 추가를 선택합니다.
- 프로젝트 이름을 추가합니다.
- 어댑터 이름을 추가합니다.
- 필요한 경우 설명을 추가합니다.

- 훈련 이미지를 가져오는 방법(매니페스트 파일, S3 버킷 또는 컴퓨터)을 선택합니다.
- 훈련 데이터를 자동 분할할지 아니면 매니페스트 파일을 가져올지 선택합니다.
- 프로젝트를 자동으로 업데이트할지 여부를 선택합니다.
- 프로젝트 생성을 클릭합니다.

AWS CLI 및 SDK를 사용하여 프로젝트를 생성하려면:

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 정보는 2단계: AWS CLI 및 AWS SDKs 설정을 참조하십시오.
2. 다음 코드를 사용하여 프로젝트를 생성합니다.

CLI

```
# Request
# Creating Content Moderation Project
aws rekognition create-project \
    --project-name "project-name" \
    --feature CONTENT_MODERATION \
    --auto-update ENABLED
    --profile profile-name
```

프로젝트 설명

DescribeProjects API를 사용하여 프로젝트에 연결된 모든 어댑터에 대한 정보를 포함하여 프로젝트에 대한 정보를 가져올 수 있습니다.

AWS CLI 및 SDK를 사용하여 프로젝트를 설명하는 방법:

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 정보는 2단계: AWS CLI 및 AWS SDKs 설정을 참조하십시오.
2. 다음 코드를 사용하여 프로젝트를 설명합니다.

CLI

```
# Request
# Getting CONTENT_MODERATION project details
aws rekognition describe-projects \
    --features CONTENT_MODERATION
    --profile profile-name
```

프로젝트 삭제

Rekognition 콘솔을 사용하거나 DeleteProject API를 직접 호출하여 프로젝트를 삭제할 수 있습니다. 프로젝트를 삭제하려면 먼저 연결된 어댑터를 모두 삭제해야 합니다. 삭제된 프로젝트 또는 모델은 복구할 수 없습니다.

AWS 콘솔을 사용하여 프로젝트를 삭제하려면:

- Rekognition 콘솔에 로그인합니다.
- 사용자 지정 조절을 클릭합니다.
- 프로젝트와 연결된 모든 어댑터를 삭제해야 프로젝트 자체를 삭제할 수 있습니다. 어댑터를 선택한 다음 삭제를 선택하여 프로젝트와 연결된 어댑터를 모두 삭제합니다.
- 프로젝트를 선택한 다음 삭제 버튼을 선택합니다.

AWS CLI 및 SDK를 사용하여 프로젝트를 삭제하려면:

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 정보는 2단계: AWS CLI 및 AWS SDKs 설정을 참조하십시오.
2. 다음 코드를 사용하여 프로젝트를 삭제합니다.

CLI

```
aws rekognition delete-project
  --project-arn project_arn \
  --profile profile-name
```

프로젝트 버전 생성

CreateProjectVersion 작업을 사용하여 배포할 어댑터를 훈련시킬 수 있습니다. CreateProjectVersion은 먼저 프로젝트와 연결된 어댑터의 새 버전을 생성한 다음 어댑터 훈련을 시작합니다. CreateProjectVersion에서 보내는 응답은 모델 버전에 대한 Amazon 리소스 이름(ARN)입니다. 훈련을 완료하는 데 다소 시간이 걸립니다. DescribeProjectVersions를 직접 호출하여 현재 상태를 가져올 수 있습니다. 모델을 훈련할 때 Rekognition은 프로젝트와 연결된 훈련 및 테스트 데이터 세트를 사용합니다. 콘솔을 사용하여 데이터 세트를 생성합니다. 자세한 내용은 데이터 세트에 관한 섹션을 참조하세요.

Rekognition 콘솔을 사용하여 프로젝트 버전을 생성하려면

- AWS Rekognition 콘솔에 로그인합니다.
- 사용자 지정 조절을 클릭합니다.
- 프로젝트를 선택합니다.
- “프로젝트 세부 정보” 페이지에서 어댑터 생성을 선택합니다.
- “프로젝트 생성” 페이지에서 프로젝트 세부 정보, 교육 이미지, 테스트 이미지에 필요한 세부 정보를 입력한 다음 프로젝트 생성을 선택합니다.
- “이미지에 레이블 할당” 페이지에서 이미지에 레이블을 추가하고 완료되면 훈련 시작을 선택합니다.

AWS CLI 및 SDK를 사용하여 프로젝트 버전을 생성하려면:

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 정보는 2단계: AWS CLI 및 AWS SDKs 설정을 참조하십시오.
2. 다음 코드를 사용하여 프로젝트 버전을 생성합니다.

CLI

```
# Request
aws rekognition create-project-version \
 --project-arn project-arn \
 --training-data '{Assets=[GroundTruthManifest={S3Object="amzn-s3-demo-source-
bucket",Name="manifest.json"}]}' \
 --output-config S3Bucket=amzn-s3-demo-destination-bucket,S3KeyPrefix=my-results \
 --feature-config "ContentModeration={ConfidenceThreshold=70}"
 --profile profile-name
```

프로젝트 버전 설명

DescribeProjectVersions 작업을 사용하여 프로젝트와 연결된 어댑터를 나열하고 설명할 수 있습니다. ProjectVersionArns에서 최대 10개의 모델 버전을 지정할 수 있습니다. 값을 지정하지 않으면 프로젝트의 모든 모델 버전에 대한 설명을 반환합니다.

AWS CLI 및 SDK를 사용하여 프로젝트 버전을 설명하는 방법:

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 정보는 2단계: AWS CLI 및 AWS SDKs 설정을 참조하십시오.
2. 다음 코드를 사용하여 프로젝트 버전을 설명합니다.

CLI

```
aws rekognition describe-project-versions
  --project-arn project_arn \
  --version-names [versions]
```

프로젝트 버전 삭제

DeleteProjectVersion 작업을 사용하여 프로젝트와 연결된 Rekognition 어댑터를 삭제할 수 있습니다. 실행 중이거나 훈련 중인 어댑터는 삭제할 수 없습니다. 어댑터의 상태를 확인하려면 DescribeProjectVersions 작업을 직접 호출하고 해당 작업이 반환한 상태 필드를 확인합니다. 실행 중인 어댑터를 중지하려면 StopProjectVersion을 직접 호출하세요. 모델이 훈련 중인 경우 훈련이 완료될 때까지 기다렸다가 삭제하세요. 프로젝트와 연결된 모든 어댑터를 삭제해야 프로젝트 자체를 삭제할 수 있습니다.

Rekognition 콘솔을 사용하여 프로젝트 버전을 삭제하려면

- Rekognition 콘솔에 로그인합니다.
- 사용자 지정 조절을 클릭합니다.
- 프로젝트 탭에서 모든 프로젝트와 연결된 어댑터를 볼 수 있습니다. 어댑터를 선택한 다음 삭제를 선택합니다.

AWS CLI 및 SDK를 사용하여 프로젝트 버전을 삭제하려면:

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 정보는 2단계: AWS CLI 및 AWS SDKs 설정을 참조하십시오.
2. 다음 코드를 사용하여 프로젝트 버전을 삭제합니다.

CLI

```
# Request
aws rekognition delete-project-version
  --project-version-arn model_arn \
  --profile profile-name
```

사용자 지정 조절 어댑터 자습서

이 자습서에서는 Rekognition 콘솔을 사용하여 어댑터를 생성, 훈련, 평가, 사용 및 관리하는 방법을 보여줍니다. AWS SDK를 사용하여 어댑터를 생성, 사용 및 관리하려면 섹션을 참조하세요AWS CLI 및 SDKs를 사용하여 어댑터 관리.

어댑터를 사용하면 여러분의 필요와 사용 사례에 맞게 모델 행동을 사용자 지정하여 Rekognition API 작업의 정확성을 높일 수 있습니다. 이 자습서를 통해 어댑터를 생성한 후에는 DetectModerationLabels와 같은 작업을 통해 자체 이미지를 분석하고 향후 개선을 위해 어댑터를 재훈련할 때 이를 사용할 수 있습니다.

이 자습서에서는 다음을 수행하는 방법을 알아봅니다.

- Rekognition 콘솔을 사용하여 프로젝트 생성
- 훈련 데이터에 주석 달기
- 훈련 데이터 세트를 기반으로 어댑터 훈련
- 어댑터 성능 검토
- 어댑터를 사용하여 이미지 분석

사전 조건

이 튜토리얼을 완료하기 전에 어댑터 생성 및 사용를 끝까지 읽어보는 것을 권장합니다.

어댑터를 만들려면 Rekognition 콘솔 도구를 사용하여 프로젝트를 만들고 자체 이미지를 업로드하고 주석을 추가한 다음 이러한 이미지를 기반으로 어댑터를 학습시킬 수 있습니다. 시작하려면 프로젝트 생성 및 어댑터 훈련 섹션을 참조하세요.

또는 Rekognition의 콘솔 또는 API를 사용하여 이미지에 대한 예측을 검색한 다음 예측을 검증하고 이러한 예측에 대해 어댑터를 훈련할 수 있습니다. 시작하려면 대량 분석, 예측 검증, 어댑터 훈련 섹션을 참조하세요.

이미지 주석

Rekognition 콘솔로 이미지에 레이블을 지정하여 이미지에 직접 주석을 달거나 Rekognition 대량 분석을 사용하여 이미지에 주석을 추가한 다음 레이블이 올바르게 지정되었는지 확인할 수 있습니다. 시작하려면 아래 주제 중 하나를 선택하세요.

주제

- 프로젝트 생성 및 어댑터 훈련
- 대량 분석, 예측 검증, 어댑터 훈련

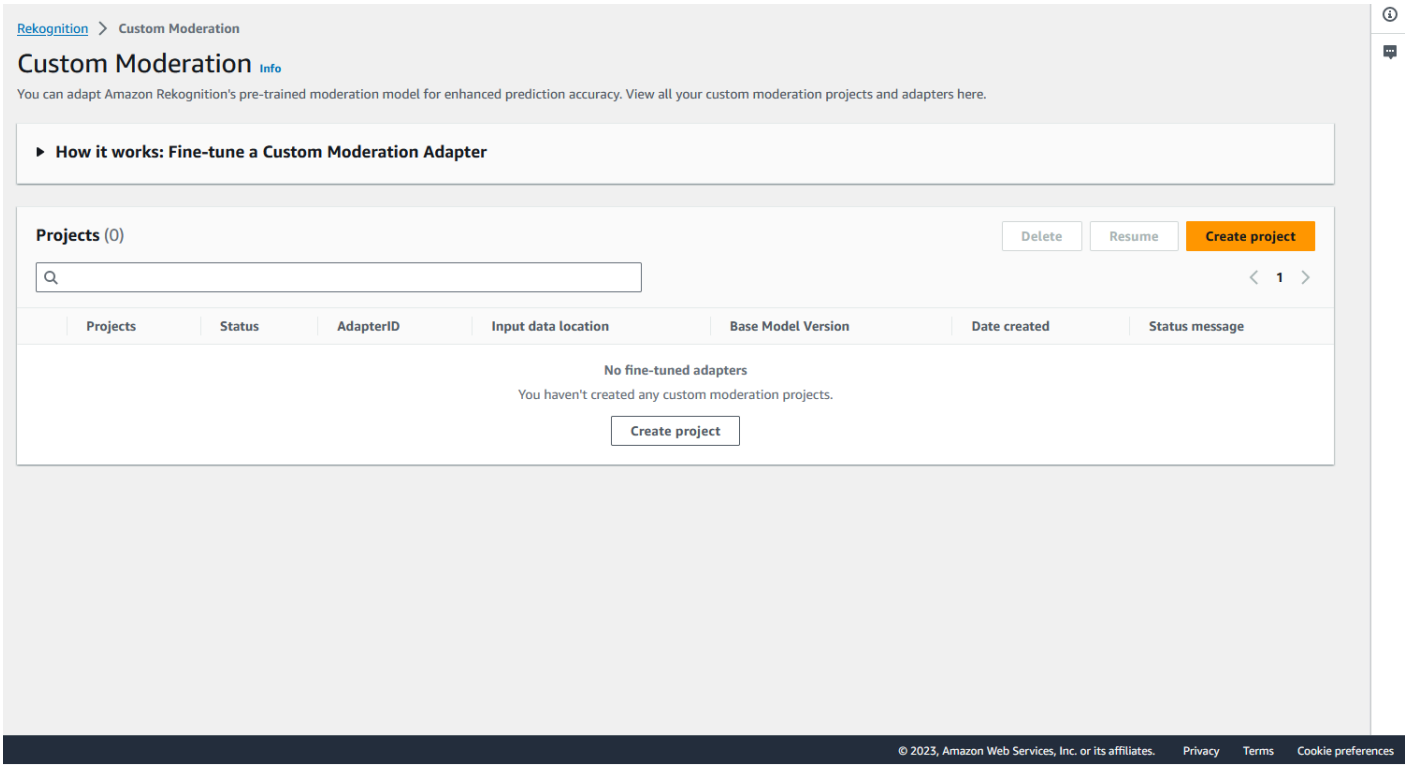
프로젝트 생성 및 어댑터 훈련

Rekognition 콘솔을 사용하여 이미지에 주석을 추가함으로써 어댑터를 훈련시키려면 다음 단계를 수행합니다.

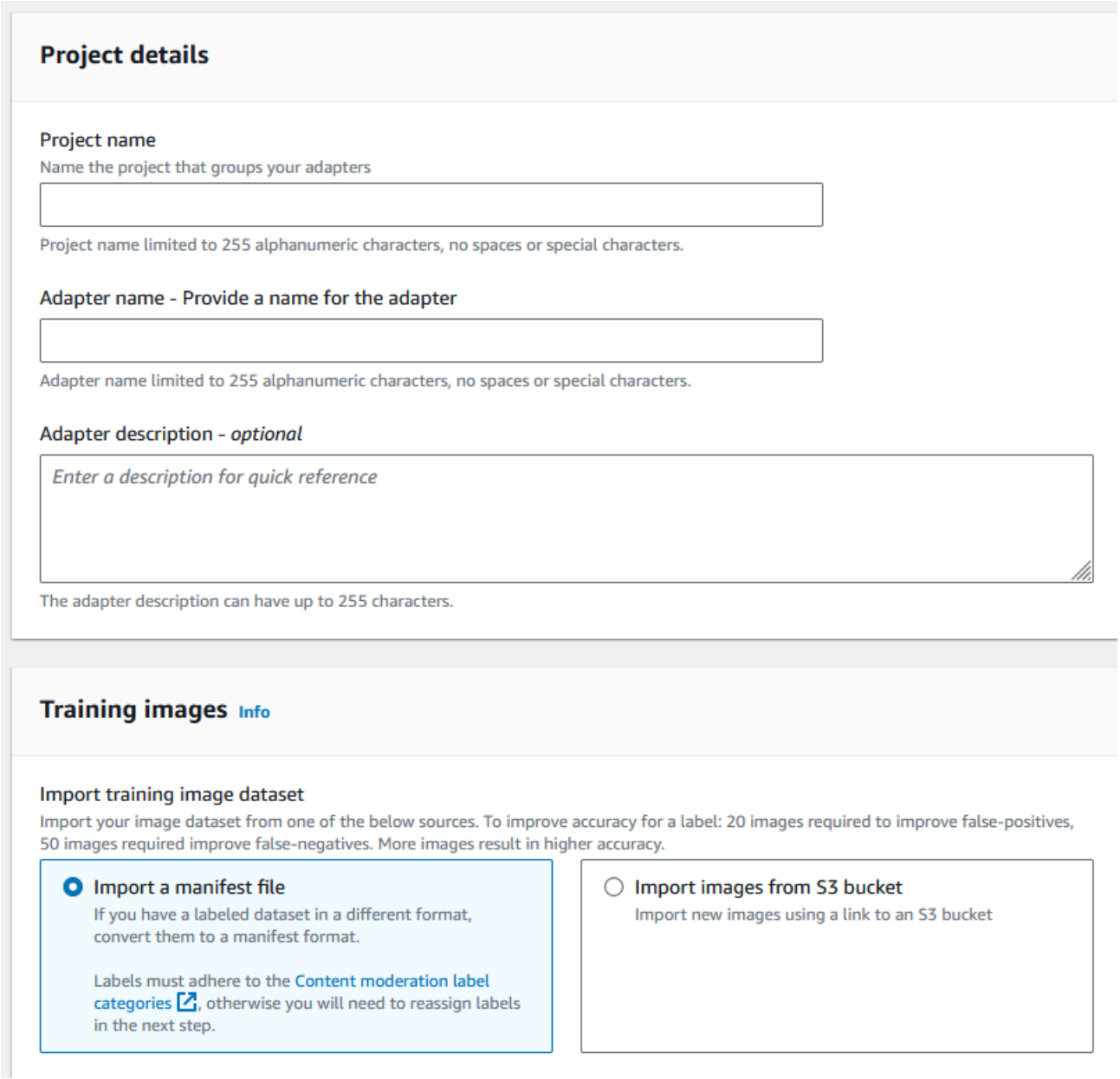
프로젝트 만들기

어댑터를 훈련시키거나 사용하려면 먼저 어댑터를 넣을 프로젝트를 만들어야 합니다. 어댑터 훈련에 사용되는 이미지도 제공해야 합니다. 프로젝트, 어댑터, 이미지 데이터 세트를 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/rekognition/ Rekognition 콘솔을 엽니다.
2. 왼쪽 창에서 사용자 지정 조절을 선택합니다. Rekognition 사용자 지정 조절 랜딩 페이지가 표시됩니다.



3. 사용자 지정 조절 랜딩 페이지에는 모든 프로젝트 및 어댑터 목록이 표시되며 어댑터를 만들 수 있는 버튼도 있습니다. 새 프로젝트와 어댑터를 생성하려면 프로젝트 생성을 선택합니다.
4. 어댑터를 처음 생성하는 경우 프로젝트 및 어댑터와 관련된 파일을 저장할 Amazon S3 버킷을 생성하라는 메시지가 표시됩니다. Amazon S3 버킷 생성을 선택합니다.
5. 다음 페이지에서 어댑터 이름과 프로젝트 이름을 입력합니다. 원하는 경우 어댑터 설명을 제공하세요.



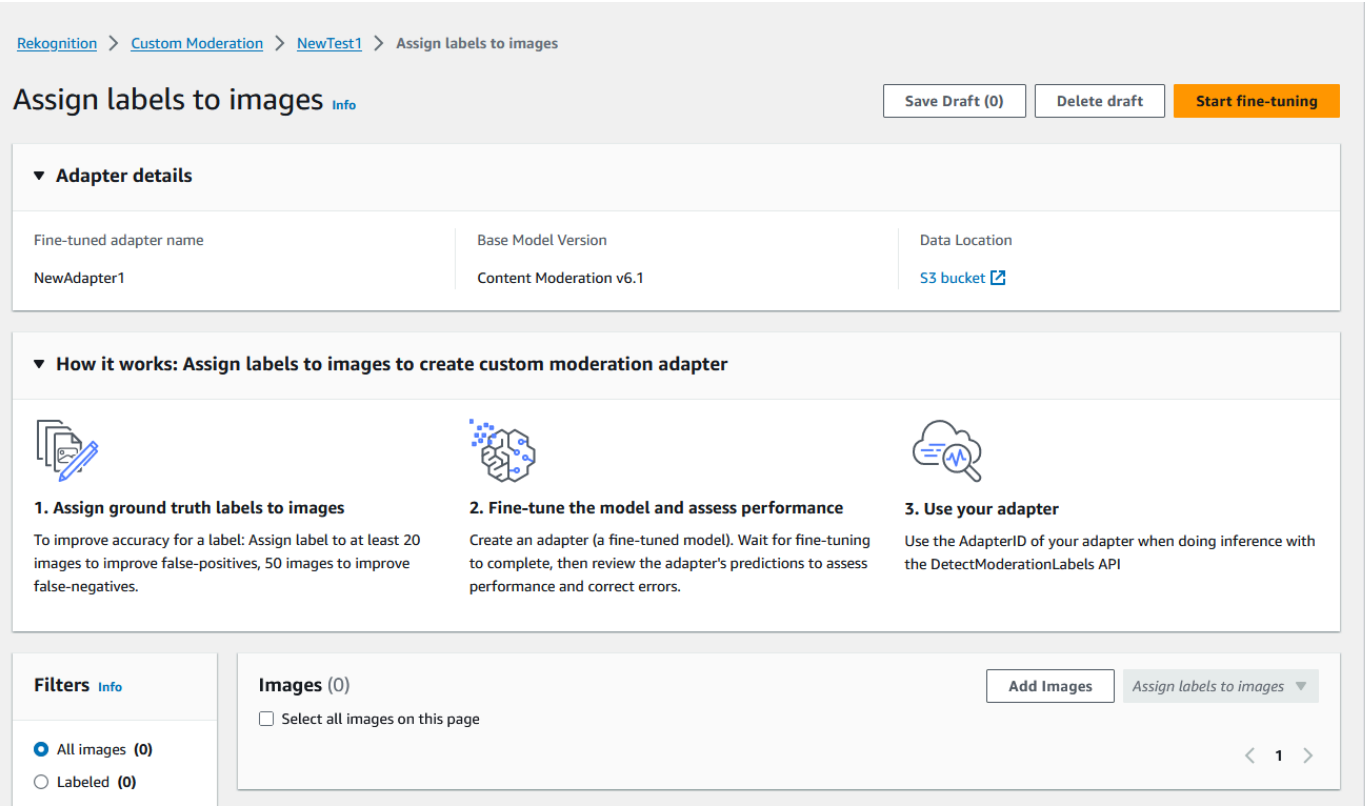
6. 어댑터의 이미지도 이 단계에서 제공합니다. 컴퓨터에서 이미지 가져오기, 매니페스트 파일 가져오기, 또는 Amazon S3 버킷에서 이미지 가져오기 중에서 선택할 수 있습니다. Amazon S3 버킷에서 이미지를 가져오려는 경우 훈련 이미지가 포함된 버킷과 폴더의 경로를 제공하세요. 컴퓨터에서 직접 이미지를 업로드하는 경우 한 번에 최대 30개의 이미지만 업로드할 수 있다는 점에 유의하세요. 주석이 포함된 매니페스트 파일을 사용하는 경우 이미지 주석 달기에 관련된 아래 단계를 건너뛰고 어댑터 성능 검토에 관한 다음 섹션으로 진행해도 됩니다.
7. 테스트 데이터 세트 세부 정보 섹션에서 자동 분할을 선택하여 Rekognition에서 테스트 데이터로 적절한 비율의 이미지를 자동으로 선택하도록 하거나 수동으로 매니페스트 파일 가져오기를 선택할 수 있습니다.

8. 이 정보를 입력한 후 프로젝트 생성을 선택합니다.

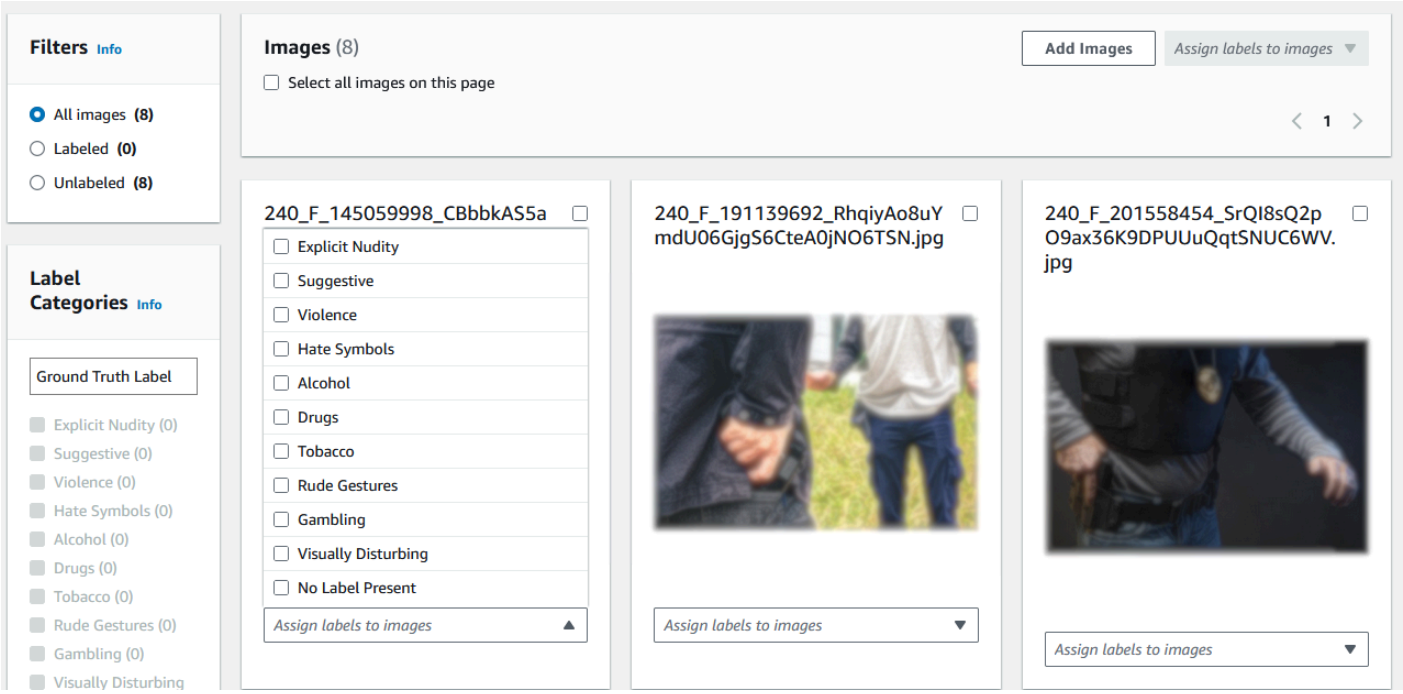
어댑터 훈련

주석이 없는 자체 이미지로 어댑터를 훈련시키려면

1. 어댑터가 포함된 프로젝트를 선택한 다음 이미지에 레이블 할당 옵션을 선택합니다.
2. 이미지에 레이블 할당 페이지에서 훈련 이미지로 업로드된 모든 이미지를 볼 수 있습니다. 왼쪽에 있는 두 개의 속성 선택 패널을 사용하여 레이블이 지정되거나 레이블이 지정되지 않은 상태 및 레이블 카테고리별로 이미지를 필터링할 수 있습니다. 이미지 추가 버튼을 선택하여 훈련 데이터 세트에 이미지를 더 추가할 수 있습니다.



3. 훈련 데이터 세트에 이미지를 추가한 후에는 이미지에 레이블을 주석으로 달아야 합니다. 이미지를 업로드하면 업로드한 이미지가 표시되도록 “이미지에 레이블 할당” 페이지가 업데이트됩니다. Rekognition Moderation에서 지원하는 레이블의 드롭다운 목록에서 이미지에 적합한 레이블을 선택하라는 메시지가 표시됩니다. 레이블을 두 개 이상 선택할 수 있습니다.
4. 훈련 데이터의 모든 이미지에 레이블을 추가할 때까지 이 프로세스를 계속하세요.
5. 모든 데이터에 레이블을 지정한 후 훈련 시작을 선택하여 모델 훈련을 시작하면 어댑터가 생성됩니다.



6. 훈련 프로세스를 시작하기 전에 원하는 태그를 어댑터에 추가할 수 있습니다. 어댑터에 사용자 지정 암호화 키를 제공하거나 AWS KMS 키를 사용할 수도 있습니다. 원하는 태그를 추가하고 원하는 대로 암호화를 사용자 지정한 후 어댑터 훈련을 선택하여 어댑터에 대한 훈련 프로세스를 시작하세요.
7. 어댑터가 훈련을 마칠 때까지 기다리세요. 훈련이 완료되면 어댑터 생성이 완료되었다는 알림을 받게 됩니다.

어댑터 상태가 “교육 완료”가 되면 어댑터 지표를 검토할 수 있습니다.

대량 분석, 예측 검증, 어댑터 훈련

Rekognition의 콘텐츠 조절 모델에서 대량 분석 예측을 검증하여 어댑터를 훈련시키려면 다음 단계를 완료하세요.

Rekognition의 콘텐츠 조절 모델에서 예측을 검증하여 어댑터를 훈련시키려면 다음을 수행해야 합니다.

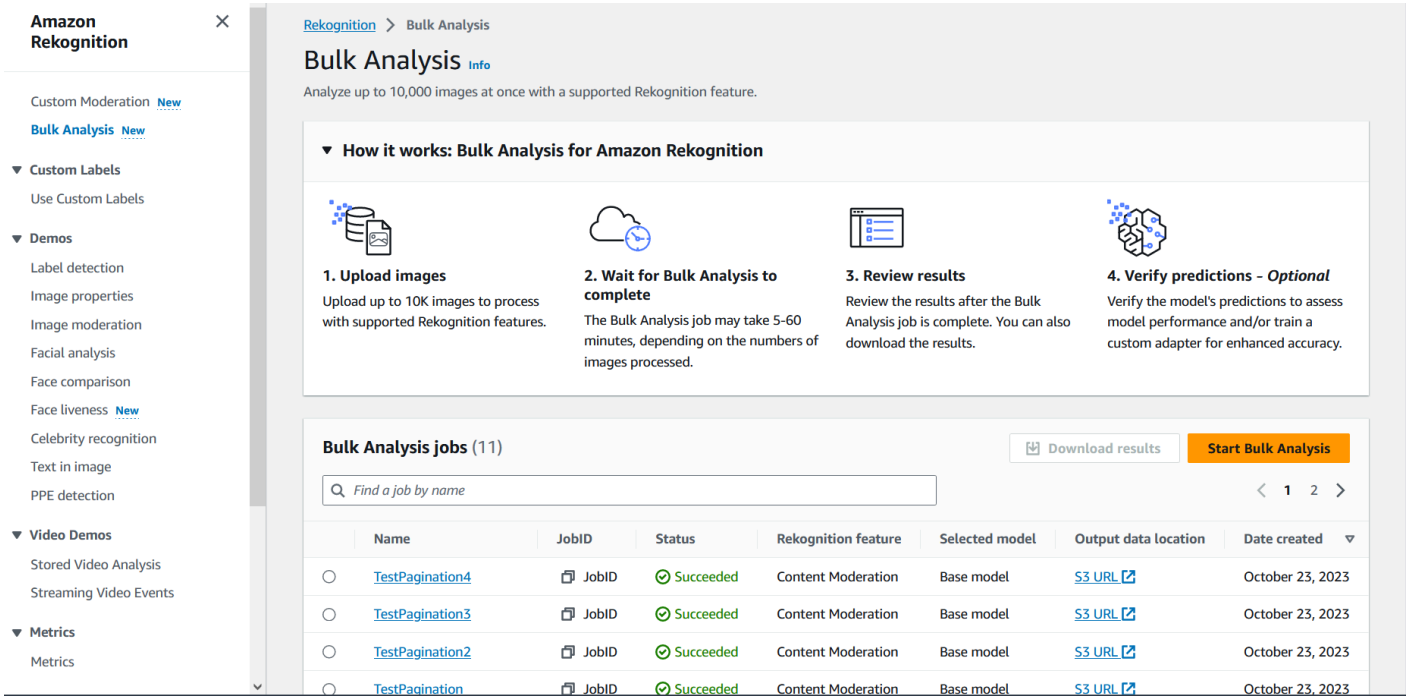
1. 이미지에 대한 대량 분석 수행
2. 이미지에 대해 반환된 예측 결과를 확인하세요.

Rekognition의 기본 모델 또는 이미 만든 어댑터를 사용해 대량 분석을 수행하여 이미지에 대한 예측을 얻을 수 있습니다.

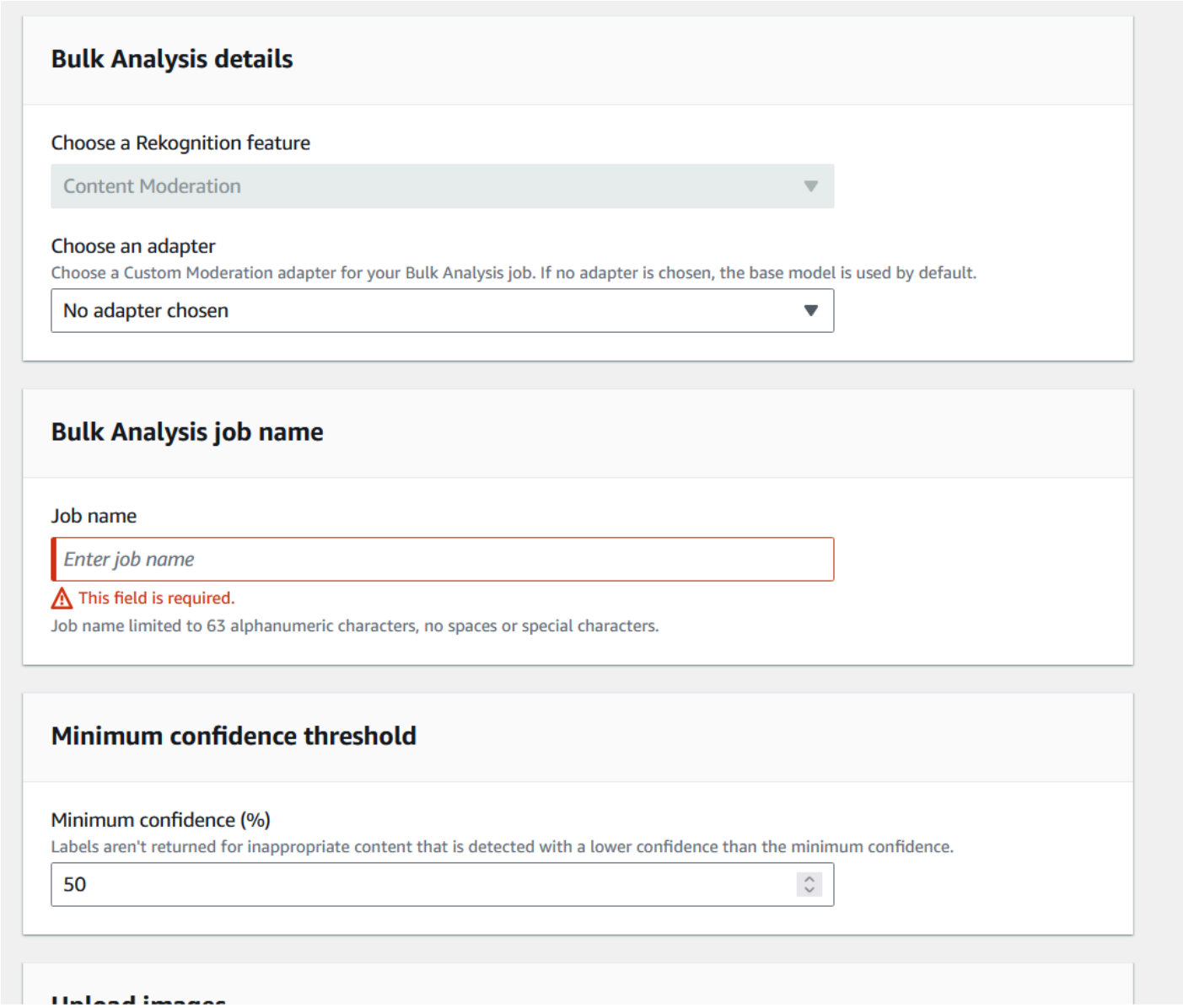
이미지에 대한 대량 분석 실행

검증한 예측에 따라 어댑터를 훈련시키려면 먼저 Rekognition의 기본 모델 또는 선택한 어댑터를 사용하여 이미지 배치를 분석하는 대량 분석 작업을 시작해야 합니다. 대량 분석 작업을 실행하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/rekognition/ Amazon Rekognition 콘솔을 엽니다.
2. 왼쪽 창에서 대량 분석을 선택합니다. 대량 분석 랜딩 페이지가 나타납니다. 대량 분석 시작을 선택합니다. 대량 분석 기능 개요는 이미지를 업로드하고, 분석을 기다리고, 결과를 검토하며, 선택적으로 모델 예측을 확인하는 단계를 보여줍니다. 기본 모델을 사용하여 콘텐츠 조절에 대한 최근 대량 분석 작업을 나열합니다.

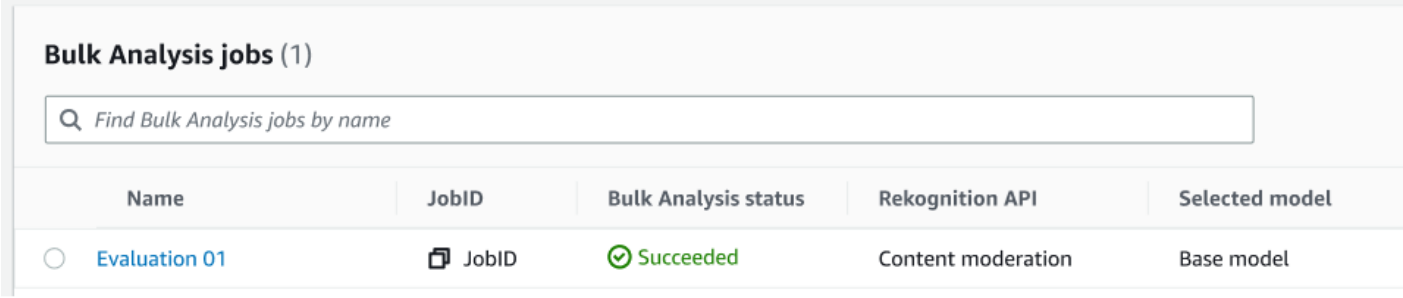


3. 어댑터를 처음 생성하는 경우 프로젝트 및 어댑터와 관련된 파일을 저장할 Amazon Simple Storage Service 버킷을 생성하라는 메시지가 표시됩니다. Amazon S3 버킷 생성을 선택합니다.
4. 어댑터 선택 드롭다운 메뉴를 사용하여 대량 분석에 사용할 어댑터를 선택합니다. 어댑터를 선택하지 않은 경우 기본적으로 기본 모델이 사용됩니다. 본 자습서에서는 어댑터를 선택하지 않습니다.

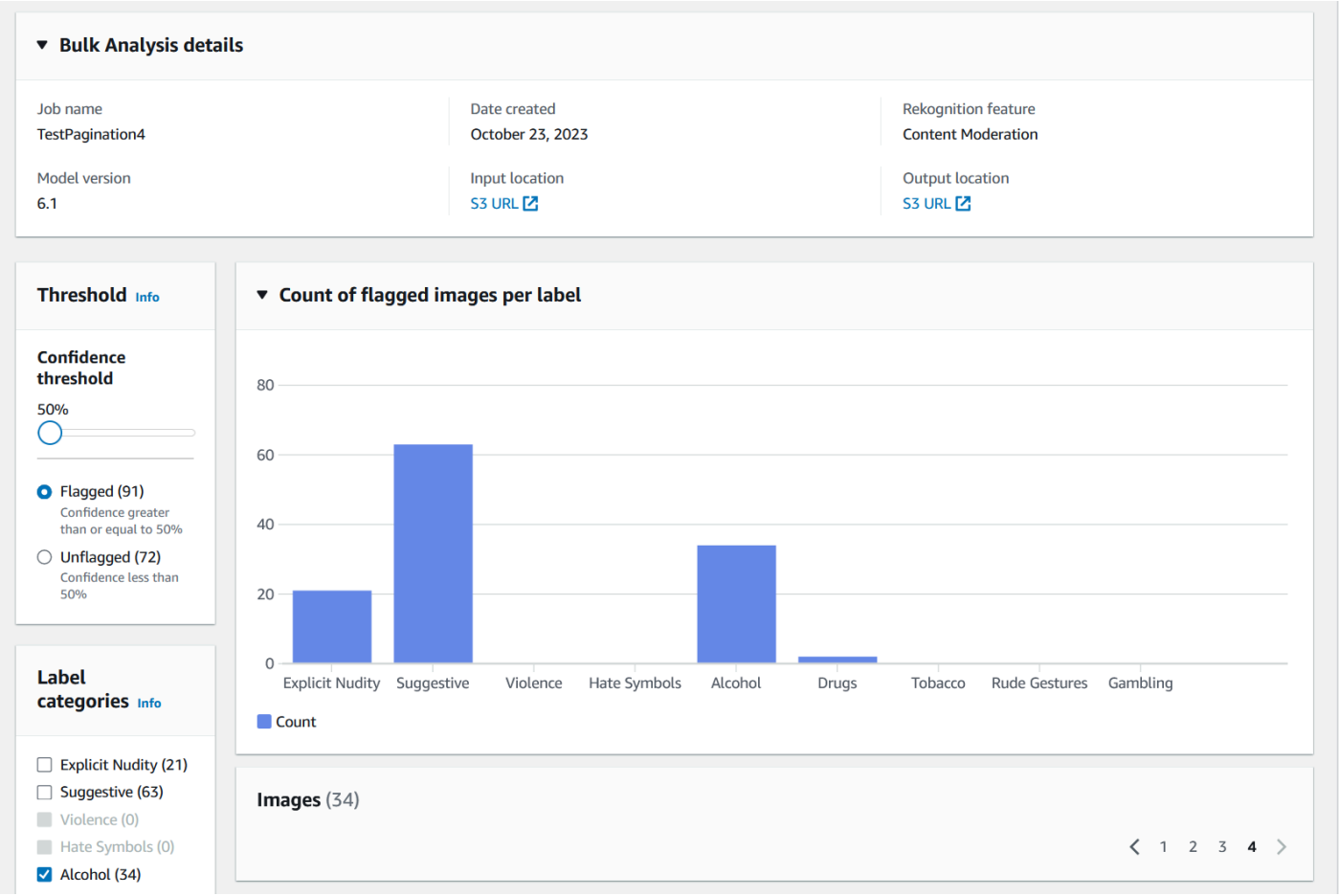


5. 벌크 분석 작업 이름 필드에 벌크 분석 작업 이름을 입력합니다.
6. 최소 신뢰도 임계값에 들어갈 값을 선택합니다. 선택한 신뢰도 임계값보다 작은 값의 레이블 예측은 반환되지 않습니다. 나중에 모델의 성능을 평가할 때 선택한 최소 신뢰도 임계값 아래로 신뢰도 임계값을 조절할 수 없다는 점에 유의하세요.
7. 대량 분석으로 분석하려는 이미지도 이 단계에서 제공합니다. 해당 이미지를 사용하여 어댑터를 훈련할 수도 있습니다. 컴퓨터에서 이미지 업로드 또는 Amazon S3 버킷에서 이미지 가져오기를 선택할 수 있습니다. Amazon S3 버킷에서 문서를 가져오려는 경우 훈련 이미지가 포함된 버킷과 폴더의 경로를 제공하세요. 컴퓨터에서 직접 문서를 업로드하는 경우 한 번에 50개의 이미지만 업로드할 수 있다는 점에 유의하세요.

8. 이 정보를 입력한 후 분석 시작을 선택합니다. 그러면 Rekognition의 기본 모델을 사용한 분석 프로세스가 시작됩니다.
9. 대량 분석 메인 페이지에서 작업의 대량 분석 상태를 확인하여 대량 분석 작업의 상태를 확인할 수 있습니다. 대량 분석 상태가 “성공”이 되면 분석 결과를 검토할 준비가 된 것입니다.



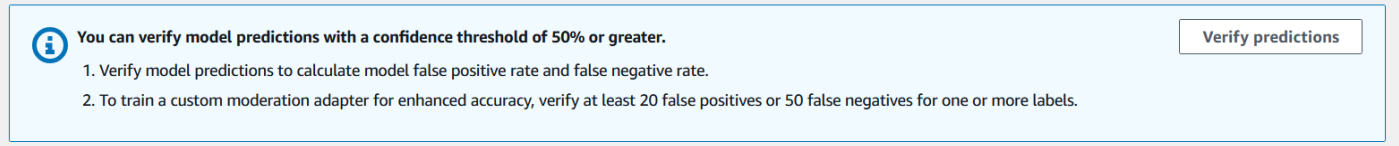
10. 대량 분석 작업 목록에서 생성한 분석을 선택합니다.
11. 대량 분석 세부 정보 페이지에서 Rekognition의 기본 모델이 업로드한 이미지에 대해 예측한 결과를 확인할 수 있습니다.
12. 기본 모델의 성능을 검토하세요. 신뢰도 임계값 슬라이더를 사용하여 이미지에 레이블을 할당하기 위해 어댑터가 가져야 하는 신뢰도 임계값을 변경할 수 있습니다. 신뢰도 임계값을 조절함에 따라 플래그 지정 인스턴스와 플래그 미지정 인스턴스의 수가 변경됩니다. 레이블 카테고리 패널에는 Rekognition에서 인식하는 최상위 카테고리가 표시되며 이 목록에서 카테고리를 선택하여 해당 레이블이 할당된 모든 이미지를 표시할 수 있습니다.



예측 검증

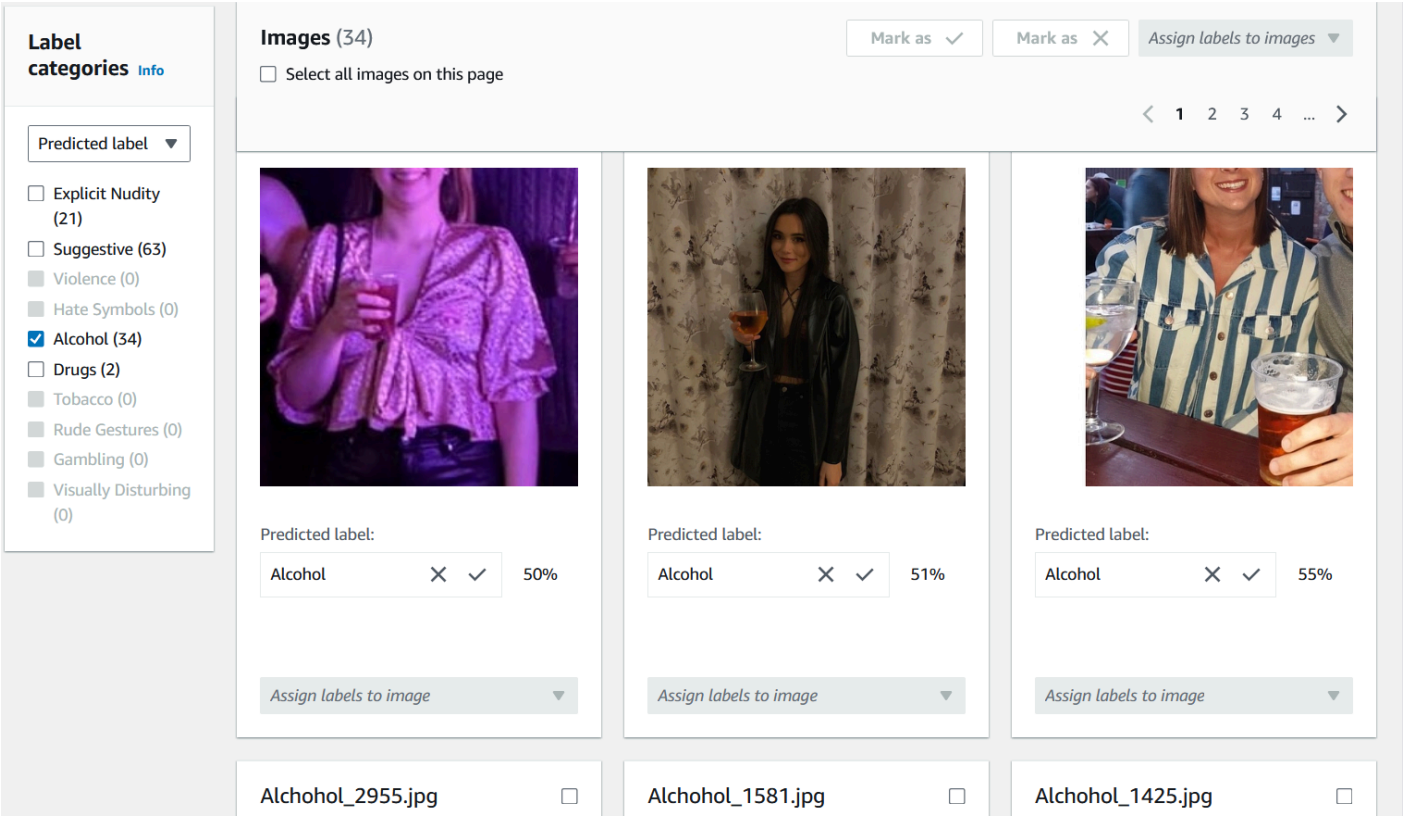
Rekognition의 기본 모델이나 선택한 어댑터의 정확도를 검토한 후 정확도를 높이고 싶다면 검증 워크플로를 활용할 수 있습니다.

1. 기본 모델 성능 검토를 마친 후에는 예측을 검증해야 합니다. 예측을 교정하여 어댑터를 훈련시킬 수 있습니다. 대량 분석 페이지 상단에서 예측 확인을 선택합니다.

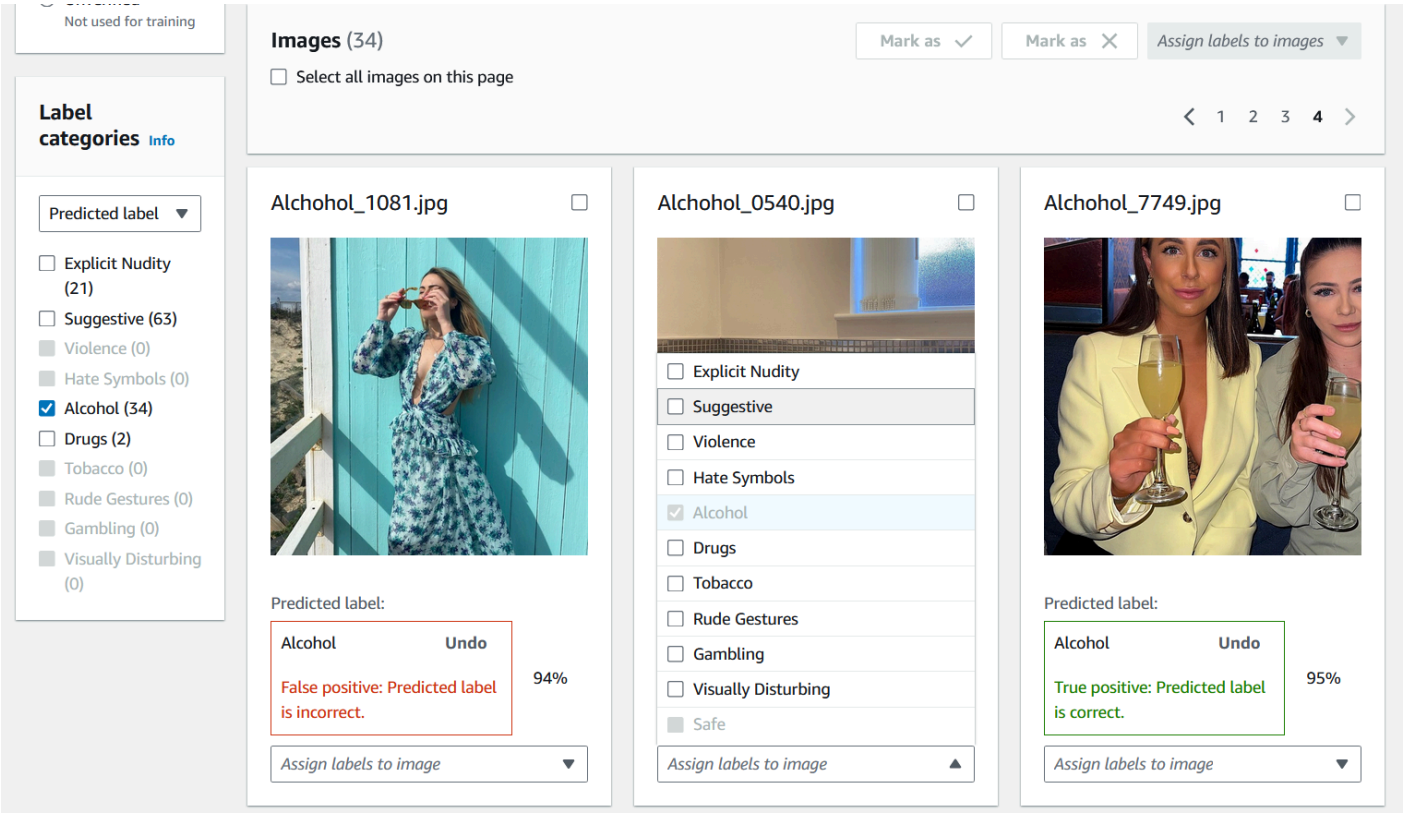


2. 예측 확인 페이지에서 Rekognition의 기본 모델 또는 선택한 어댑터에 제공한 모든 이미지를 각 이미지의 예측된 레이블과 함께 볼 수 있습니다. 이미지 아래에 있는 버튼을 사용하여 각 예측이 맞는지 틀렸는지를 확인해야 합니다. 예측을 잘못된 것으로 표시하려면 “X” 버튼을 사용하고 예측을 올바른 것으로 표시하려면 체크 표시 버튼을 사용합니다. 어댑터를 훈련시키려면 주어진 레이블에 대

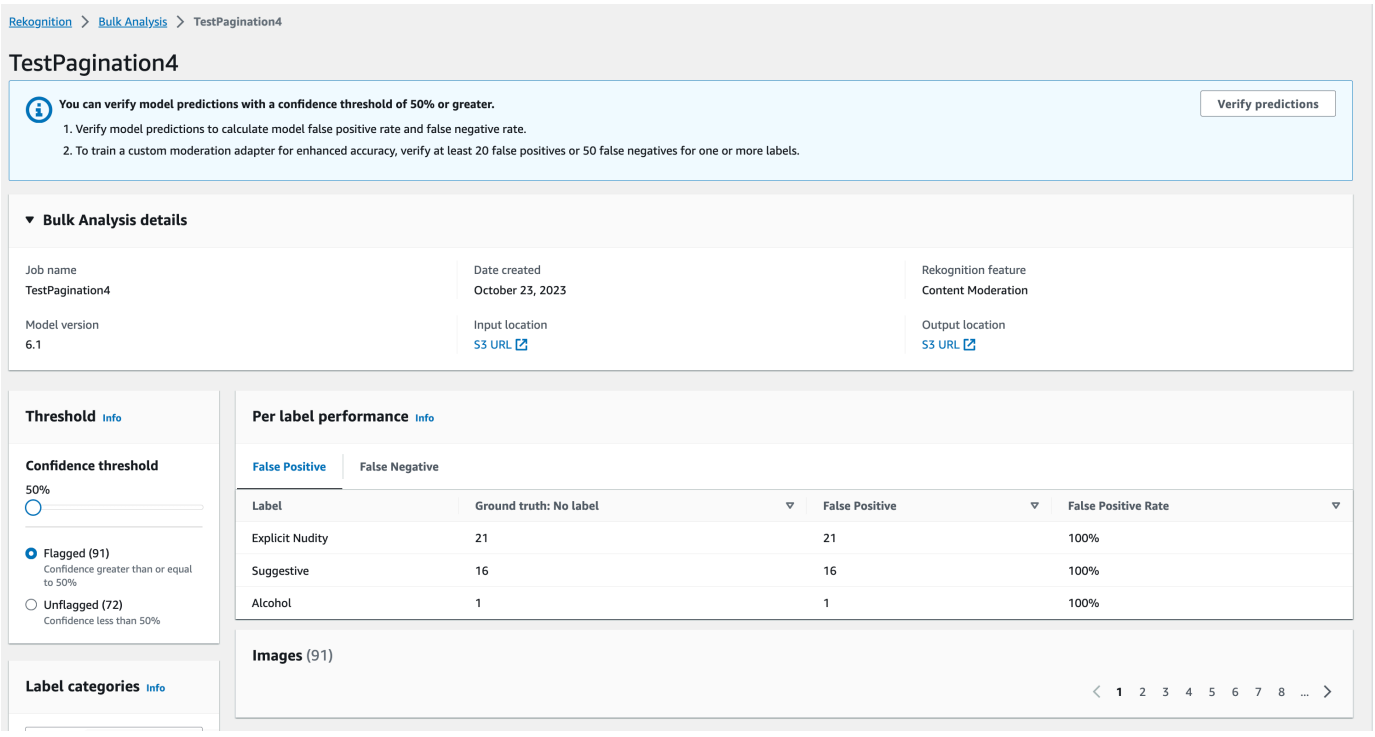
해 최소 20개의 false positive 예측과 50개의 false negative 예측을 확인해야 합니다. 더 많은 예측을 검증할수록 어댑터의 성능이 더 좋아집니다.



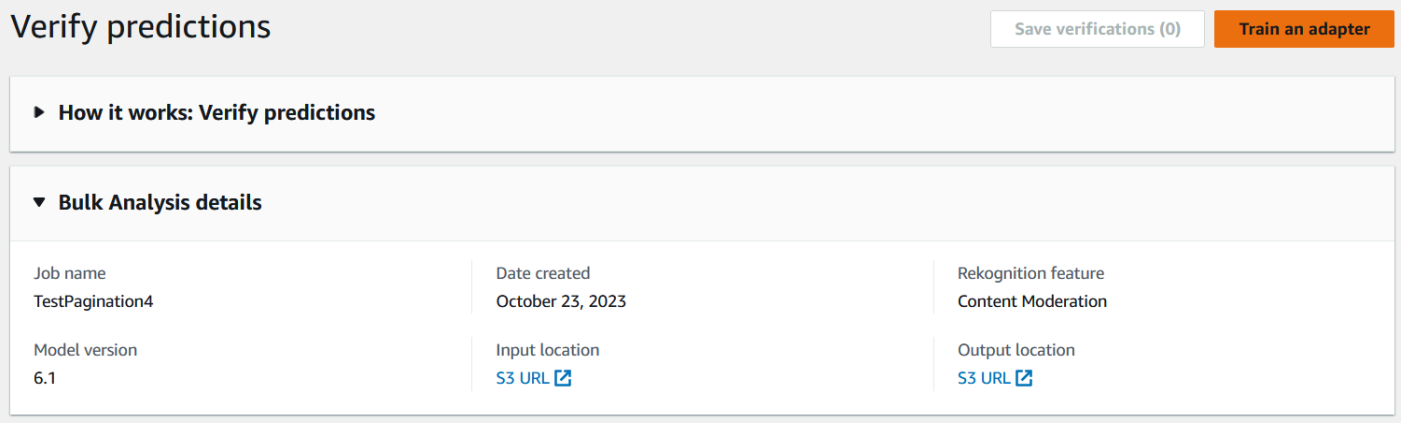
예측을 검증하면 이미지 아래의 텍스트가 변경되어 검증한 예측의 유형을 보여줍니다. 이미지를 검증한 후에는 이미지에 레이블 할당 메뉴를 사용하여 이미지에 다른 레이블을 추가할 수도 있습니다. 선택한 신뢰도 임계값에 대해 모델이 어떤 이미지에 플래그를 지정했는지 또는 플래그를 지정하지 않았는지 확인하거나 카테고리별로 이미지를 필터링할 수 있습니다.



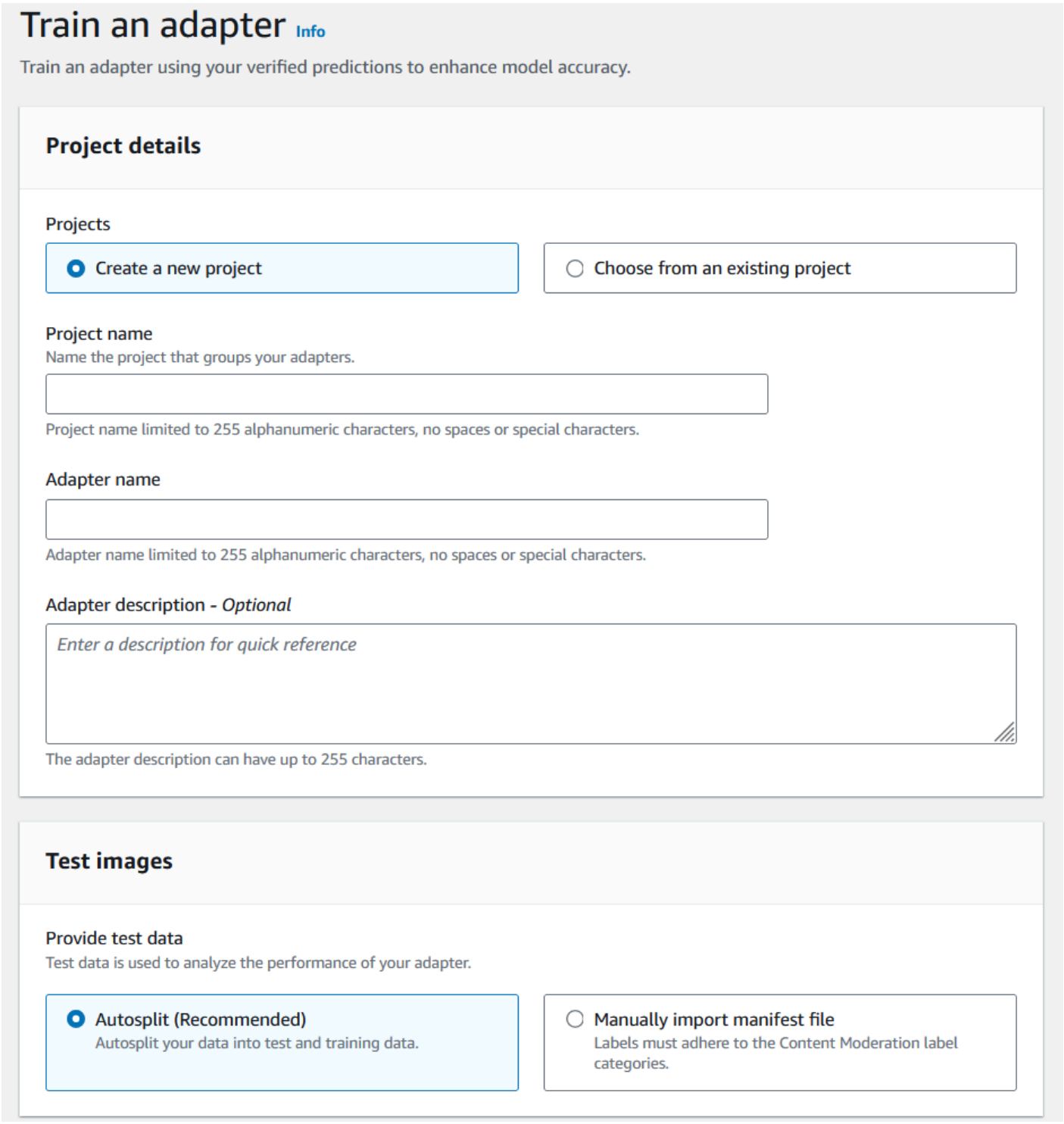
3. 확인하려는 모든 예측의 검증을 완료하면 검증 페이지의 레이블당 성능 섹션에서 검증된 예측에 대한 통계를 볼 수 있습니다. 대량 분석 세부 정보 페이지로 돌아가서 이러한 통계를 볼 수도 있습니다.



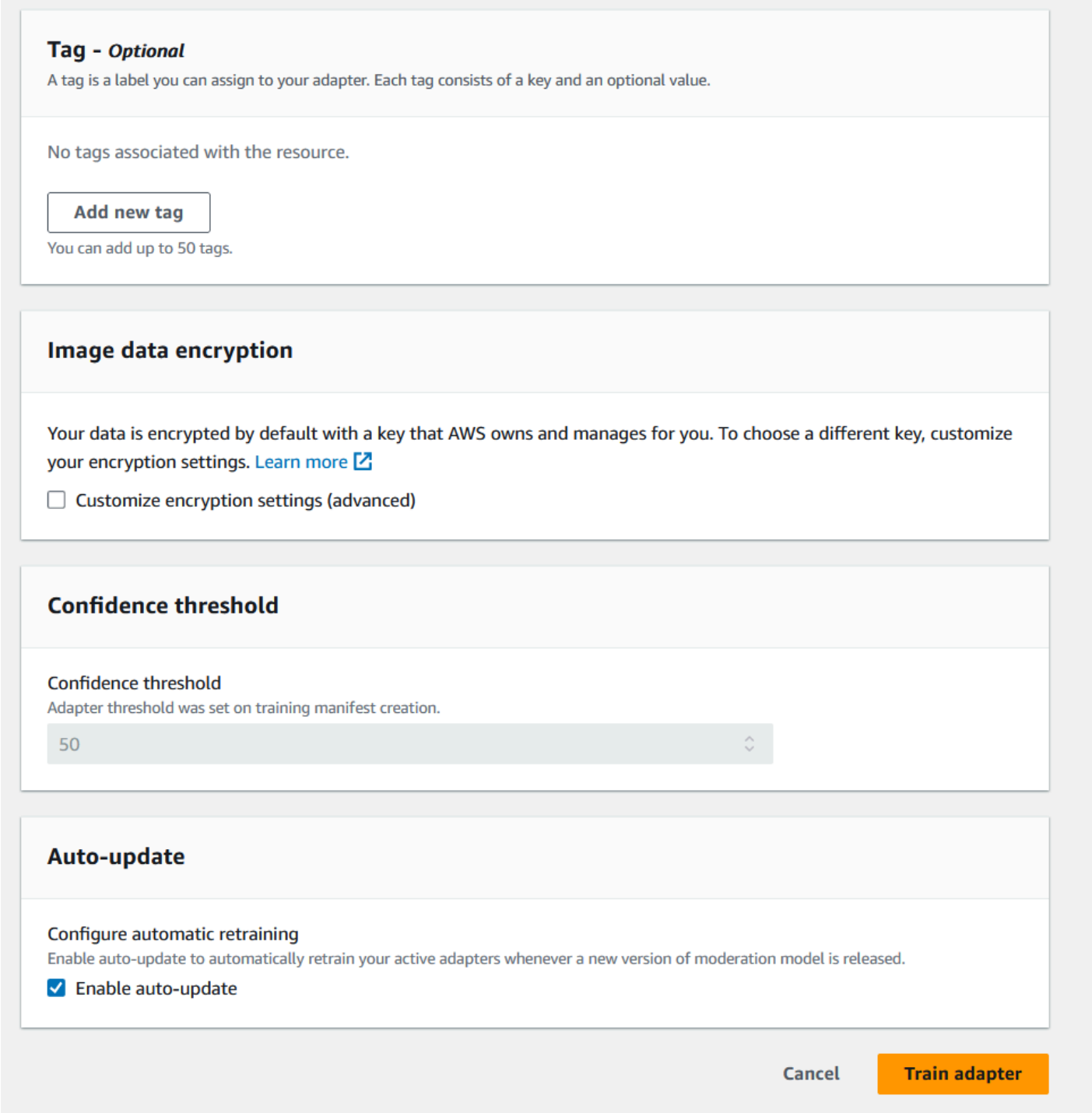
4. 레이블당 성능에 관련된 통계에 만족하면 예측 확인 페이지로 다시 이동한 다음 어댑터 훈련 버튼을 선택하여 어댑터 훈련을 시작하세요.



5. 어댑터 훈련 페이지에서는 프로젝트를 만들거나 기존 프로젝트를 선택하라는 메시지가 표시됩니다. 프로젝트와 프로젝트에 포함될 어댑터의 이름을 지정합니다. 테스트 이미지의 소스도 지정해야 합니다. 이미지를 지정할 때 Rekognition에서 자동으로 훈련 데이터의 일부를 테스트 이미지로 사용하도록 자동 분할을 선택하거나 매니페스트 파일을 수동으로 지정할 수 있습니다. 자동 분할을 선택하는 것을 권장합니다.



6. 원하는 태그를 지정하고 기본 AWS KMS 키를 사용하지 않으려면 AWS 키를 지정합니다. 자동 업데이트를 활성화한 상태로 두는 것이 좋습니다.
7. 어댑터 훈련을 선택하세요.

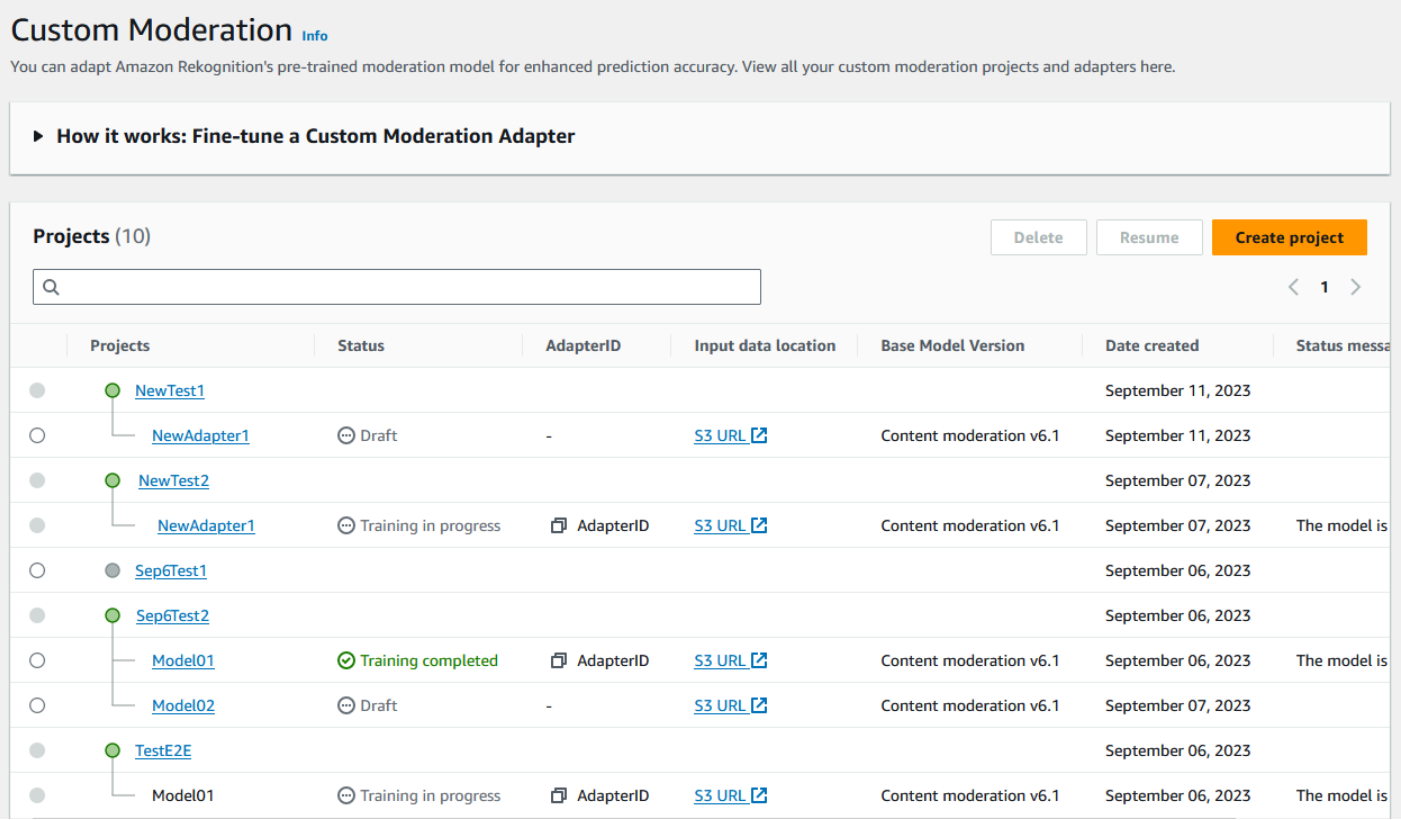


8. 사용자 지정 조절 랜딩 페이지에서 어댑터 상태가 “훈련 완료”로 표시되면 어댑터의 성능을 검토할 수 있습니다. 자세한 정보는 어댑터 성능 검토을 참조하세요.

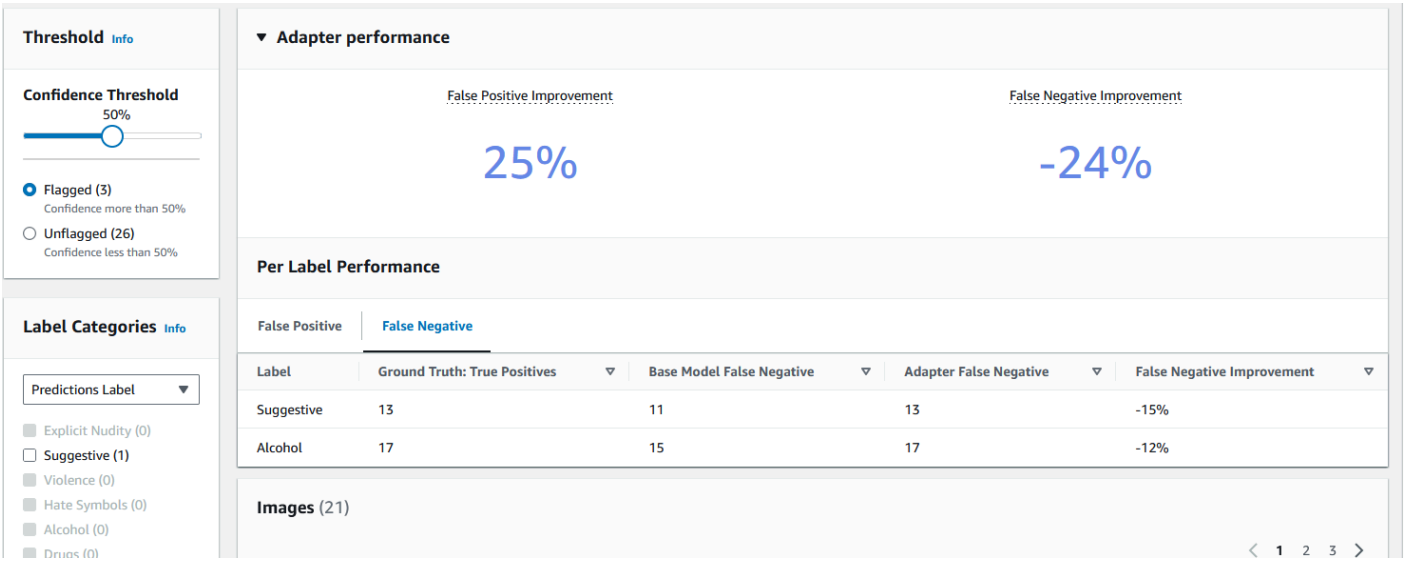
어댑터 성능 검토

어댑터 성능을 검토하려면

1. 콘솔을 사용해 사용자 지정 조절 랜딩 페이지의 프로젝트 탭에서 프로젝트와 연결된 모든 어댑터의 상태를 확인할 수 있습니다. 사용자 지정 조절 랜딩 페이지로 이동합니다.



2. 이 목록에서 검토하려는 어댑터를 선택합니다. 다음 어댑터 세부 정보 페이지에서 어댑터에 대한 다양한 지표를 볼 수 있습니다.



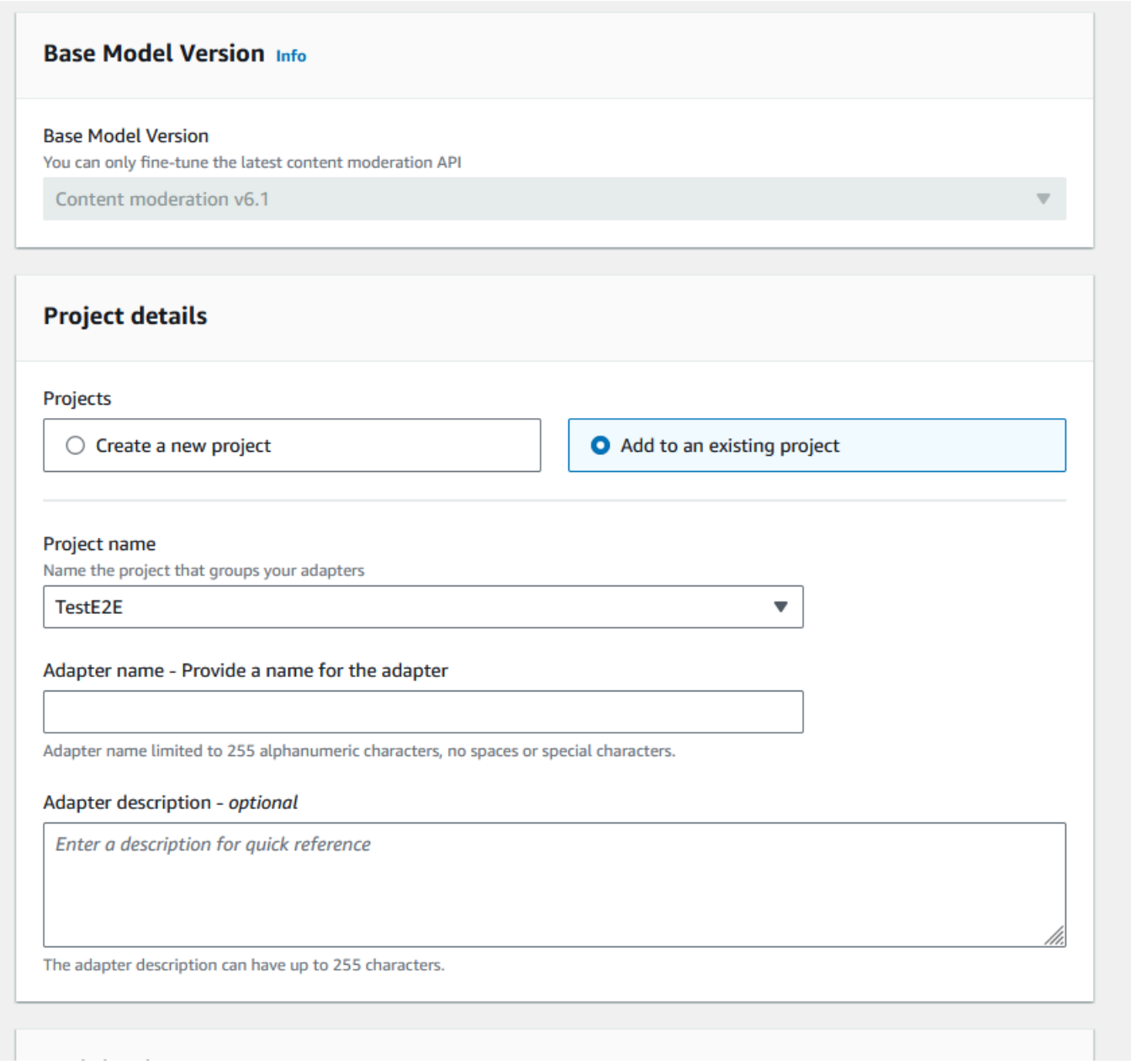
3. 임계값 패널을 사용해서 이미지에 레이블을 할당하기 위해 어댑터가 가져야 하는 신뢰도 임계값을 변경할 수 있습니다. 신뢰도 임계값을 조절함에 따라 플래그 지정 인스턴스와 플래그 미지정 인스턴

스의 수가 변경됩니다. 레이블 카테고리별로 필터링하여 선택한 카테고리의 지표를 볼 수도 있습니다. 선택한 임계값을 설정합니다.

4. 어댑터 성능 패널에서 지표를 검토하여 테스트 데이터에 대한 어댑터의 성능을 평가할 수 있습니다. 이러한 지표는 어댑터에서 추출한 결과를 테스트 세트의 “실측 정보”와 비교하여 계산됩니다.

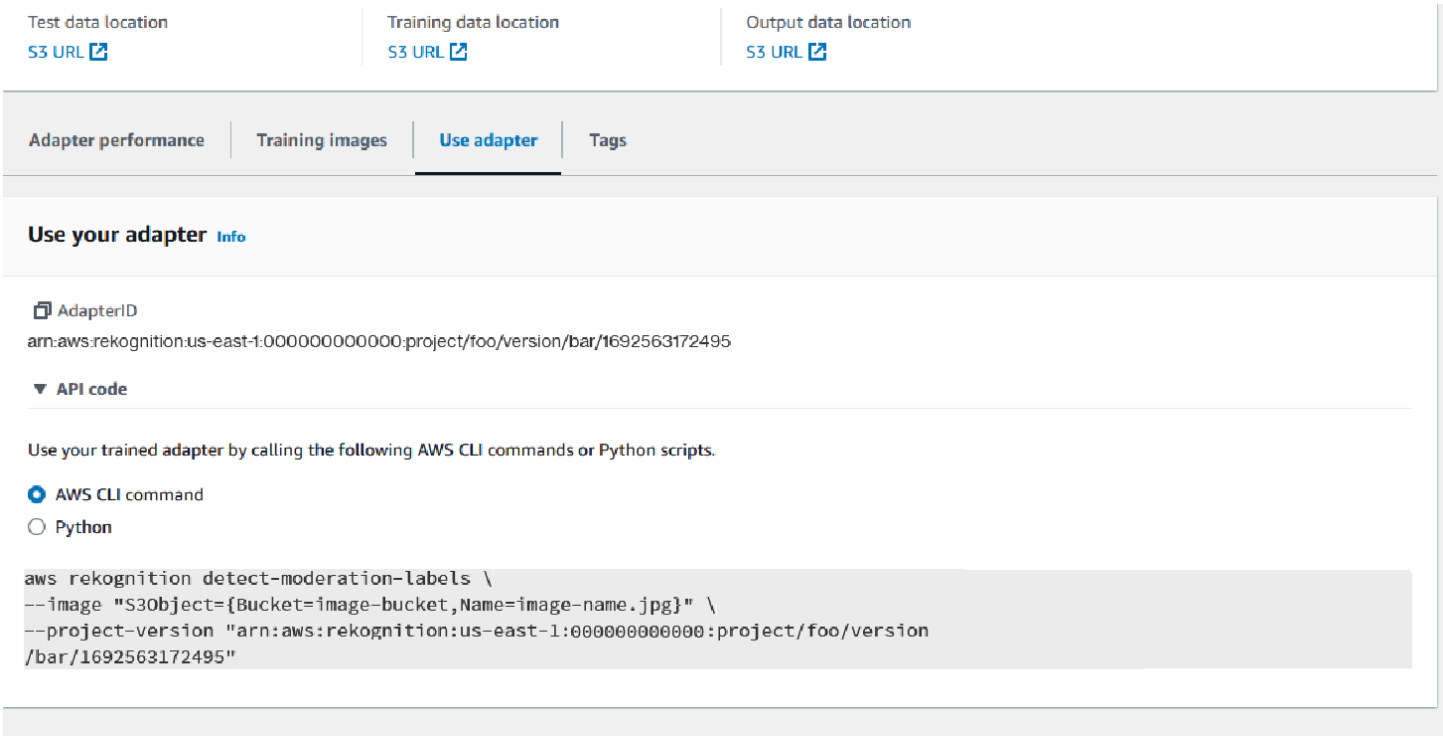
어댑터 성능 패널에는 생성한 어댑터에 대한 False Positive 개선 및 False Negative 개선률이 표시됩니다. 레이블당 성능 탭을 사용하여 각 레이블 카테고리의 어댑터 및 기본 모델 성능을 비교할 수 있습니다. 이는 기본 모델 및 어댑터의 false positive 및 false negative 예측 수를 레이블 카테고리별로 계층화하여 보여줍니다. 이러한 지표를 검토하여 어댑터의 개선이 필요한 부분을 파악할 수 있습니다. 이러한 지표에 대한 자세한 내용은 어댑터 평가 및 개선 섹션을 참조하세요.

성능을 향상시키기 위해 더 많은 훈련 이미지를 수집한 다음 프로젝트 내부를 기반으로 새 어댑터를 만들 수 있습니다. 사용자 지정 조절 랜딩 페이지로 돌아가서 프로젝트 내에 새 어댑터를 생성하여 어댑터가 훈련할 수 있도록 더 많은 훈련 이미지를 제공하기만 하면 됩니다. 이번에는 새 프로젝트 생성 대신 기존 프로젝트에 추가 옵션을 선택하고 프로젝트 이름 드롭다운 메뉴에서 새 어댑터를 만들려는 프로젝트를 선택합니다. 이전과 마찬가지로 이미지에 주석을 달거나 주석이 달린 매니페스트 파일을 제공하세요.



어댑터 사용

어댑터를 생성한 후에는 DetectModerationLabels와 같이 지원되는 Rekognition 작업에 어댑터를 제공할 수 있습니다. 어댑터로 추론을 수행하는 데 사용할 수 있는 코드 샘플을 보려면 AWS CLI와 Python 모두에 대한 코드 샘플을 볼 수 있는 "어댑터 사용" 탭을 선택합니다. 설명서에서 어댑터를 생성한 목적인 작업에 대한 섹션을 방문하여 더 많은 코드 샘플, 설정 지침 및 샘플 JSON을 볼 수도 있습니다.



어댑터 및 프로젝트 삭제

개별 어댑터를 삭제하거나 프로젝트를 삭제할 수 있습니다. 프로젝트와 연결된 모든 어댑터를 삭제해야 프로젝트 자체를 삭제할 수 있습니다.

1. 프로젝트와 연결된 어댑터를 삭제하려면 어댑터를 선택한 다음 삭제를 선택합니다.
2. 프로젝트를 삭제하려면 삭제할 프로젝트를 선택한 다음 삭제를 선택합니다.

어댑터 평가 및 개선

어댑터를 훈련할 때마다 Rekognition 콘솔 도구의 성능 지표를 검토하여 원하는 성능 수준에 어댑터가 얼마나 근접했는지 확인해야 합니다. 그런 다음 새 훈련 이미지를 업로드하고 프로젝트 내에서 새 어댑터를 훈련시켜 이미지에 대한 어댑터의 정확도를 더욱 높일 수 있습니다. 개선된 버전의 어댑터를 만든 후에는 콘솔을 사용하여 더 이상 필요하지 않은 이전 버전의 어댑터를 삭제할 수 있습니다.

DescribeProjectVersions API 작업을 사용하여 지표를 검색할 수도 있습니다.

성능 지표

훈련 프로세스를 완료하고 어댑터를 만든 후에는 어댑터가 이미지에서 정보를 얼마나 잘 추출하고 있는지 평가하는 것이 중요합니다.

Rekognition 콘솔에는 어댑터 성능 분석에 도움이 되는 두 가지 지표, 즉 false positive 개선과 false negative 개선이 제공됩니다.

콘솔의 어댑터 부분에서 “어댑터 성능” 탭을 선택하여 모든 어댑터에 대해 이러한 지표를 볼 수 있습니다. 어댑터 성능 패널에는 생성한 어댑터에 대한 False Positive 개선 및 False Negative 개선률이 표시됩니다.

False positive 개선은 기본 모델에 비해 어댑터의 false positive 인식률이 얼마나 개선되었는지를 나타냅니다. False positive 개선 값이 25%라면 어댑터가 테스트 데이터 세트에서 false positive에 대한 인식을 25% 개선했다는 의미입니다.

False negative 개선은 기본 모델에 비해 어댑터의 false negative 인식률이 얼마나 개선되었는지를 나타냅니다. False negative 개선 값이 25%라면 어댑터가 테스트 데이터 세트에서 false negative에 대한 인식을 25% 개선했다는 의미입니다.

레이블당 성능 탭을 사용하여 각 레이블 카테고리의 어댑터 및 기본 모델 성능을 비교할 수 있습니다. 이는 기본 모델 및 어댑터의 false positive 및 false negative 예측 수를 레이블 카테고리별로 계층화하여 보여줍니다. 이러한 지표를 검토하여 어댑터의 개선이 필요한 부분을 파악할 수 있습니다.

예를 들어 알코올 레이블 카테고리의 기본 모델 False Negative 비율이 15이고 어댑터의 False Negative 비율이 15 이상이라면 새 어댑터를 만들 때 알코올 레이블이 포함된 이미지를 더 추가하는 데 집중해야 한다는 것을 알 수 있습니다.

Rekognition API 작업을 사용하는 경우 DescribeProjectVersions 작업을 직접 호출하면 F1-Score 지표가 반환됩니다.

모델 개선

어댑터 배포는 반복적인 프로세스로, 목표 정확도 수준에 도달하려면 어댑터를 여러 번 훈련해야 할 수 있습니다. 어댑터를 만들고 훈련시킨 후에는 다양한 유형의 레이블에서 어댑터의 성능을 테스트하고 평가해야 합니다.

어댑터의 정확도가 어느 부분에서든 부족한 경우 해당 이미지의 새로운 예시를 추가하여 해당 레이블에 대한 어댑터의 성능을 높이세요. 어댑터가 어려움을 겪는 사례들을 반영해서 다양한 예시를 추가로 제공하세요. 어댑터에 특징적이고 다양한 이미지를 제공하면 다채로운 현실 사례를 처리할 수 있습니다.

훈련 세트에 새 이미지를 추가하고 어댑터를 다시 훈련시킨 다음 테스트 세트와 레이블을 가지고 다시 평가하세요. 어댑터가 원하는 성능 수준에 도달할 때까지 이 프로세스를 반복합니다. 특징적인 이미지와 주석을 더 많이 제공하면 false positive 및 false negative 점수가 반복되는 훈련을 통해 점차 개선될 것입니다.

매니페스트 파일 형식

다음 섹션에는 입력, 출력 및 평가 파일의 매니페스트 파일 형식 샘플이 나와 있습니다.

입력 매니페스트

매니페스트 파일은 json 줄로 구분된 파일로, 각 줄에는 단일 이미지에 대한 정보가 들어 있는 JSON이 들어 있습니다.

입력 매니페스트의 각 항목은 Amazon S3 버킷에 든 이미지 경로가 있는 `source-ref` 필드를 포함해야 하며, 사용자 지정 조절의 경우 실측 정보 주석이 있는 `content-moderation-groundtruth` 필드를 포함해야 합니다. 한 데이터 세트의 모든 이미지는 동일한 버킷에 있어야 합니다. 이 구조는 훈련 매니페스트 파일과 테스트 매니페스트 파일 모두에 공통으로 적용됩니다.

사용자 지정 조절의 `CreateProjectVersion` 작업은 입력 매니페스트에 제공된 정보를 사용하여 어댑터를 훈련시킵니다.

다음 예제는 안전하지 않은 단일 클래스가 포함된 단일 이미지에 대한 매니페스트 파일의 한 줄입니다.

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            }
        ]
   }
}
```

다음 예제는 안전하지 않은 여러 클래스, 특히 나체 및 무례한 제스처를 포함하는 안전하지 않은 단일 이미지에 대한 매니페스트 파일의 한 줄입니다.

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            },
            {
```

```
                "Name": "Nudity"
            }
        ]
    }
}
```

다음 예제는 안전하지 않은 클래스가 포함되지 않은 단일 이미지에 대한 매니페스트 파일의 한 줄입니다.

```
{
    "source-ref": "s3://foo/bar/1.jpg",
    "content-moderation-groundtruth": {
         "ModerationLabels": []
    }
}
```

지원되는 레이블의 전체 목록은 콘텐츠 조절을 참조하세요.

출력 매니페스트

훈련 작업이 완료되면 출력 매니페스트 파일이 반환됩니다. 출력 매니페스트 파일은 JSON 줄로 구분된 파일로, 각 줄에는 단일 이미지에 대한 정보가 들어 있는 JSON이 들어 있습니다. OutputManifest의 Amazon S3 경로는 `DescribeProjectVersion` 응답에서 확인할 수 있습니다.

- 데이터 세트 훈련용 `TrainingDataResult.Output.Assets[0].GroundTruthManifest.S3Object`
- 데이터 세트 테스트용 `TestingDataResult.Output.Assets[0].GroundTruthManifest.S3Object`

출력 매니페스트의 각 항목에 대해 다음 정보가 반환됩니다.

Key Name	Description
`source-ref`	Reference to an image in s3 that was provided in the input maniefst
`content-moderation-groundtruth`	Ground truth annotations that were provided in the input manifest

`detect-moderation-labels`	Adapter predictions, part of the testing dataset only
`detect-moderation-labels-base-model`	Base model predictions, part of the testing dataset only

어댑터 및 기본 모델 예측은 ConfidenceTrehsold 5.0에서 DetectModerationLabels 응답과 유사한 형식으로 반환됩니다.

다음 예제는 어댑터 및 기본 모델 예측의 구조를 보여줍니다.

```
{
   "ModerationLabels": [
      {
         "Confidence": number,
         "Name": "string",
         "ParentName": "string"
      }
   ],
   "ModerationModelVersion": "string",
   "ProjectVersion": "string"
}
```

반환되는 레이블의 전체 목록은 콘텐츠 조절을 참조하세요.

평가 결과 매니페스트

훈련 작업이 완료되면 평가 결과 매니페스트 파일이 반환됩니다. 평가 결과 매니페스트는 훈련 작업에서 출력한 JSON 파일이며, 여기에는 테스트 데이터에서 어댑터가 얼마나 좋은 결과를 냈는지에 대한 정보가 들어 있습니다.

평가 결과 매니페스트의 Amazon S3 경로는 DescribeProejctVersion 응답의 `EvaluationResult.Summary.S3Object` 필드에서 확인할 수 있습니다.

다음 예제는 평가 결과 매니페스트 구조를 보여줍니다.

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

평가 매니페스트 파일에는 다음이 포함됩니다.

- F1Score에 정의된 집계 결과
- ProjectVersionArn, 훈련 이미지의 수, 테스트 이미지의 수, 어댑터가 훈련된 레이블 등 평가 작업에 대한 세부 정보
- 기본 모델 및 어댑터 성능 둘 다에 대한 TruePositive, TrueNegative, FalsePositive, FalseNegative 결과 집계치
- 입력 신뢰도 임계값에서 계산된 기본 모델 및 어댑터 성능 모두에 대한 레이블별 TruePositive, TrueNegative, FalsePositive 및 FalseNegative 결과
- 서로 다른 신뢰도 임계값에서 기본 모델 및 어댑터 성능 모두에 대한 레이블별 TruePositive, TrueNegative, FalsePositive 및 FalseNegative 결과 집계치 신뢰도 임계값의 범위는 5에서 100까지이며 5씩 조절 가능합니다.

어댑터 훈련 모범 사례

어댑터를 생성, 훈련, 사용할 때 다음 모범 사례를 따르는 것을 권장합니다.

1. 샘플 이미지 데이터는 고객이 억제하려는 전형적인 오류를 담고 있어야 합니다. 모델이 시각적으로 유사한 이미지에서 실수를 반복하는 경우 해당 이미지를 대량으로 훈련해야 합니다.
2. 특정 조절 레이블에 대해 모델이 잘못 인식하는 이미지만 가져오는 대신 모델이 올바른 예측을 하는 이미지도 입력하세요.
3. 훈련용으로는 최소 50개의 False Negative 샘플 또는 20개의 False Positive 샘플을 제공하고 테스트용으로는 최소 20개의 샘플을 제공하세요. 그러나 어댑터 성능을 높이려면 주석이 달린 이미지를 최대한 많이 제공하는 것이 좋습니다.
4. 모든 이미지에 대해 중요한 모든 레이블에 주석 달기 - 이미지에 표시된 레이블에 대해 주석을 달아야 하는 경우, 다른 모든 이미지에서 이 레이블과 일치하는 부분에 주석을 달아야 합니다.
5. 샘플 이미지 데이터에는 프로덕션 환경에서 분석될 이미지의 특징적인 사례에 초점을 맞추어 레이블의 변형을 최대한 많이 포함해야 합니다.

자동 업데이트 권한 설정

Rekognition은 사용자 지정 어댑터에 대한 자동 업데이트 기능을 지원합니다. 즉, 프로젝트에 AutoUpdate 플래그가 활성화되어 있는 경우 자동 재훈련을 최대한 활용할 수 있다는 것입니다. 이러한 자동 업데이트에는 훈련/테스트 데이터 세트와 고객 어댑터를 훈련시키는 AWS KMS 키에 액세스할 수 있는 권한이 필요합니다. 아래 단계에 따라 이러한 권한을 제공할 수 있습니다.

Amazon S3 버킷 권한

기본적으로 모든 Amazon S3 버킷 및 객체는 프라이빗입니다. 버킷을 생성한 AWS 계정인 리소스 소유자만 버킷과 버킷에 포함된 모든 객체에 액세스할 수 있습니다. 그러나 리소스 소유자는 버킷 정책을 작성함으로써 다른 리소스 및 사용자에게 액세스 권한을 부여할 수 있습니다.

사용자 지정 어댑터 훈련에 있어 입력 데이터 세트 및 훈련 결과의 원본으로 사용할 Amazon S3 버킷을 생성하거나 수정하려는 경우 버킷 정책을 추가로 수정해야 합니다. Amazon S3 버킷에서 읽거나 쓰려면 Rekognition에 다음 권한이 있어야 합니다.

Rekognition에 필요한 Amazon S3 정책

Rekognition에는 다음과 같은 속성을 가진 권한 정책이 필요합니다.

- Statement SID
- 버킷 이름
- Rekognition의 서비스 보안 주체 이름
- Rekognition, 버킷 및 이의 모든 콘텐츠에 필요한 리소스
- Rekognition에서 해야 하는 작업

다음 정책은 Rekognition에서 자동 재훈련 중에 Amazon S3 버킷에 액세스하도록 허용합니다.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "AllowRekognitionAutoUpdateActions",
            "Principal": {
                "Service": "rekognition.amazonaws.com"
            },
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:HeadObject",
                "s3:HeadBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
```

```
            ]
        }
    ]
}
```

이 안내서에 따라 위의 버킷 정책을 S3 버킷에 추가할 수 있습니다.

버킷 정책에 대한 자세한 내용은 여기를 참조하세요.

AWS KMS 키 권한

Rekognition을 사용하면 사용자 지정 어댑터를 훈련하는 동안 KmsKeyId를 선택적으로 제공할 수 있습니다. Rekognition은 이 키를 사용하여 모델 훈련을 위해 서비스에 복사되는 훈련 및 테스트 이미지를 암호화합니다. 이 키는 출력 Amazon S3 버킷(OutputConfig)에 기록된 훈련 결과 및 매니페스트 파일을 암호화하는 데에도 사용됩니다.

KMS 키를 사용자 지정 어댑터 훈련에 대한 입력으로 제공하기로 선택한 경우(즉 `Rekognition:CreateProjectVersion`) Rekognition 서비스 주체가 향후 자동 재훈련에 이 키를 사용할 수 있도록 KMS 키 정책을 추가로 수정해야 합니다. Rekognition에는 다음 권한이 반드시 필요합니다.

Rekognition 필수 AWS KMS 키 정책

Amazon Rekognition에는 다음과 같은 속성을 가진 권한 정책이 필요합니다.

- Statement SID
- Amazon Rekognition의 서비스 보안 주체 이름
- Amazon Rekognition에서 해야 하는 작업

다음 키 정책은 Amazon Rekognition이 자동 재훈련 중에 Amazon KMS 키에 액세스할 수 있도록 허용합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "KeyPermissions",
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "rekognition.amazonaws.com"
            },
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

이 가이드에 따라 AWS KMS 키에 위의 AWS KMS 정책을 추가할 수 있습니다.

AWS KMS 정책에 대한 자세한 내용은 여기에서 확인하세요.

AWS Rekognition에 대한 Health Dashboard 인증

AWS 상태 대시보드는 Rekognition에서 보내는 알림에 대한 지원을 제공합니다. 이러한 알림은 애플리케이션에 영향을 미칠 수 있는 Rekognition 모델의 예정된 변경 사항에 대한 안내 및 조치를 위한 지침을 제공합니다. 현재는 Rekognition 콘텐츠 조절 기능과 관련된 이벤트만 사용할 수 있습니다.

AWS 상태 대시보드는 AWS 상태 서비스의 일부입니다. 설정이 필요하지 않으며, 계정에서 인증된 사용자면 누구나 볼 수 있습니다. 자세한 내용은 AWS Health Dashboard 시작을 참조하세요.

다음 메시지와 유사한 메시지를 받으면 조치를 취하라는 경보로 받아들여야 합니다.

알림 예시: Rekognition 콘텐츠 조절에 새 모델 버전을 사용할 수 있습니다.

Rekognition은 AWS 상태 대시보드에 `AWS_MODERATION_MODEL_VERSION_UPDATE_NOTIFICATION` 이벤트를 게시하여 조정 모델의 새 버전이 릴리스되었음을 나타냅니다. DetectModerationLabels API 및 어댑터를 해당 API와 함께 사용하는 경우 이 이벤트는 중요합니다. 새 모델은 사용 사례에 따라 품질에 영향을 미칠 수 있으며 결국에는 이전 모델 버전을 대체하게 될 것이기 때문입니다. 이 알림을 받으면 모델 품질을 검증하고 모델 업데이트의 일정을 파악하는 것이 좋습니다.

모델 버전 업데이트 알림을 받은 경우 조치를 취하라는 경보로 취급해야 합니다. 어댑터를 사용하지 않는 경우 기존 사용 사례를 가지고 업데이트된 모델의 품질을 평가해야 합니다. 어댑터를 사용하는 경우 업데이트된 모델로 새 어댑터를 훈련시키고 품질을 평가해야 합니다. 자동 훈련 세트를 사용하는 경우 새 어댑터가 자동으로 훈련될 것이고 그 후 훈련된 어댑터의 품질을 평가할 수 있습니다.

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

EventBridge하려면 Amazon EventBridge를 사용하여 AWS AWS 상태 이벤트 모니터링을 참조하세요. EventBridge

Amazon Augmented AI를 사용한 부적절한 콘텐츠 검토

Amazon Augmented AI(Amazon A2I)를 사용하면 기계 학습 예측의 인적 검토에 필요한 워크플로우를 구축할 수 있습니다.

Amazon Rekognition은 Amazon A2I와 직접 통합되어 안전하지 않은 이미지를 감지하는 사용 사례에 대한 인적 검토를 쉽게 구현할 수 있습니다. Amazon A2I는 이미지 조정을 위한 인적 검토 워크플로우를 제공합니다. 이를 통해 Amazon Rekognition의 예측을 쉽게 검토할 수 있습니다. 사용 사례에 대한 신뢰도 임계값을 정의하고 시간이 지남에 따라 조정할 수 있습니다. Amazon A2I를 이용하면 조직 또는 Amazon Mechanical Turk 내에서 검토자 풀을 사용할 수 있습니다. 품질과 보안 절차 준수를 위해 AWS에서 사전 검열을 거친 공급업체 인력도 이용할 수 있습니다.

다음 단계에서는 Amazon Rekognition을 이용해 Amazon A2I를 설정하는 방법을 안내합니다. 먼저, 인적 검토를 트리거하는 조건이 있는 Amazon A2I를 사용해 흐름 정의를 생성합니다. 그런 다음 흐름 정의의 Amazon 리소스 이름(ARN)을 Amazon Rekognition `DetectModerationLabel` 작업에 전달합니다. `DetectModerationLabel` 응답에서 인적 검토가 필요한지 여부를 확인할 수 있습니다. 인적 검토의 결과는 흐름 정의에 의해 설정된 Amazon S3 버킷에서 사용할 수 있습니다.

Amazon Rekognition과 함께 Amazon A2I를 사용하는 방법에 대한 종합적인 데모를 보려면 Amazon SageMaker AI 개발자 안내서에서 다음 자습서 중 하나를 참조하세요.

- 데모: Amazon A2I 콘솔에서 시작
- 데모: Amazon A2I API를 사용하여 시작

  API 사용을 시작하기 위해 예제 Jupyter Notebook을 실행할 수도 있습니다. SageMaker AI 노트북 인스턴스에서 Amazon Augmented AI(Amazon A2I)와 Amazon Rekognition 통합 [예제] 노트북을 사용하려면 Amazon A2I Jupyter Notebook으로 SageMaker 노트북 인스턴스 사용을 참조하세요.

Amazon A2I에서 DetectModerationLabels 실행

> **Note**
>
> 동일한 AWS 리전에서 모든 Amazon A2I 및 Amazon Rekognition 리소스를 생성합니다.

1. SageMaker AI 설명서의 Amazon Augmented AI 시작하기에 나열된 사전 요구 사항을 완료하세요.

   또한 SageMaker AI 설명서의 Amazon Augmented AI에서의 권한 및 보안 페이지에서와 같이 IAM 권한을 설정해야 합니다.
2. SageMaker AI 설명서의 인적 검토 워크플로 생성에 관한 지침을 따르세요.

인적 검토 워크플로우는 이미지 처리를 관리합니다. 여기에는 인적 검토를 트리거하는 조건, 이미지가 전송되는 작업 팀, 작업 팀이 사용하는 UI 템플릿 및 작업 팀의 결과가 전송되는 Amazon S3 버킷이 포함됩니다.

`CreateFlowDefinition` 호출 내에서 `HumanLoopRequestSource`를 "AWS/Rekognition/DetectModerationLabels/Image/V3"으로 설정해야 합니다. 그런 다음 인적 검토를 트리거하는 조건을 설정하는 방법을 결정해야 합니다.

Amazon Rekognition에서는 `ModerationLabelConfidenceCheck` 및 `Sampling`의 두 가지 `ConditionType` 옵션을 사용할 수 있습니다.

`ModerationLabelConfidenceCheck` 는 중재 레이블의 신뢰도가 범위 내에 있을 때 인적 루프를 생성합니다. 마지막으로, `Sampling`은 인적 검토를 위해 처리된 문서의 임의의 백분율을 보냅니다. 각 `ConditionType`은 다른 `ConditionParameters` 세트를 사용하여 인적 검토 결과를 설정합니다.

`ModerationLabelConfidenceCheck` 에는 사람이 검토해야 할 키를 설정하는 `ConditionParameters` `ModerationLableName`이 있습니다. 또한 LessThan, GreaterThan 및 Equals를 사용하여 인적 검토에 보낼 백분율 범위를 설정하는 신뢰도도 있습니다. `Sampling`에는 인적 검토로 보낼 문서의 백분율을 설정하는 `RandomSamplingPercentage`이 있습니다.

다음 코드 예제는 `CreateFlowDefinition`의 부분 호출입니다. "Suggestive"(외설적 콘텐츠) 레이블에서는 98% 미만, "Female Swimwear or Underwear"(여성 수영복 또는 속옷) 레이블에는 95% 이상으로 평가된 경우 인적 검토를 위해 이미지를 보냅니다. 즉, 이미지가 외설적인 것으로 간주되지는 않지만 수영복이나 속옷을 착용한 여성이 있는 경우 인적 검토를 통해 이미지를 다시 확인할 수 있습니다.

```
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`은 `DetectModerationLabels`를 호출할 때 다음 단계에서 사용하는 `FlowDefinitionArn`을 반환합니다.

자세한 내용은 SageMaker AI API 참조의 CreateFlowDefinition을 참조하세요.

3. 부적절한 이미지 감지에 나온 것처럼 `DetectModerationLabels`를 호출할 때 `HumanLoopConfig` 파라미터를 설정합니다. `HumanLoopConfig` 세트를 사용하는 `DetectModerationLabels` 직접 호출의 예는 4단계를 참조하세요.

   a. `HumanLoopConfig` 파라미터 내에서 `FlowDefinitionArn`을 2단계에서 생성한 흐름 정의의 ARN으로 설정합니다.

   b. `HumanLoopName`을 설정합니다. 이는 지역 내에서 고유해야 하며 소문자여야 합니다.

   c. (선택 사항) `DataAttributes`를 사용하여 Amazon Rekognition에 전달한 이미지에 개인 식별 정보가 없는지 여부를 설정할 수 있습니다. Amazon Mechanical Turk에 정보를 보내려면 이 파라미터를 설정해야 합니다.

4. `DetectModerationLabels`를 실행합니다.

다음 예제에서는 AWS CLI 및를 사용하여 HumanLoopConfig 세트DetectModerationLabels로 AWS SDK for Python (Boto3) 를 실행하는 방법을 보여줍니다.

AWS SDK for Python (Boto3)

다음 요청 예제에서는 SDK for Python(Boto3)을 사용합니다. 자세한 내용은 AWS SDK for Python(Boto) API 참조의 detect_moderation_labels를 참조하세요.

```
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

다음 요청 예제에서는 AWS CLI를 사용합니다. 자세한 내용은 AWS CLI 명령 참조의 detect-moderation-labels 섹션을 참조하세요.

```
$ 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"]}}'
```

HumanLoopConfig가 활성화된 상태로 DetectModerationLabels를 실행하면 Amazon Rekognition이 SageMaker AI API 작업 StartHumanLoop를 직접 호출합니다. 이 명령은 DetectModerationLabels에서 답변을 받고 예제의 흐름 정의 조건에 대해 확인합니다. 검토 조건을 충족하는 경우 HumanLoopArn을 반환합니다. 흐름 정의에서 설정한 작업 팀의 구성원은 이제 이미지를 검토할 수 있게 되었다는 의미입니다. Amazon Augmented AI 런타임 작업을 호출하면 DescribeHumanLoop는 루프의 결과에 대한 정보를 제공합니다. 자세한 내용은 Amazon Augmented AI API 참조 설명서의 DescribeHumanLoop를 참조하세요.

이미지를 검토한 후에는 흐름 정의의 출력 경로에 지정된 버킷에서 결과를 확인할 수 있습니다. 또한 검토가 완료되면 Amazon A2I는 Amazon CloudWatch Events의 알림을 보냅니다. 어떤 이벤트를 찾아봐야 하는지 알아보려면 SageMaker AI 설명서에서 CloudWatch 이벤트를 참조하세요.

자세한 내용은 SageMaker AI 설명서에서 Amazon Augmented AI 시작하기 참조하세요.

텍스트 감지

Amazon Rekognition은 이미지와 비디오에서 얼굴을 감지할 수 있습니다. 그런 다음 감지된 텍스트를 기계 판독 가능한 텍스트로 변환할 수 있습니다. 이미지에서 기계가 읽을 수 있는 텍스트 감지 기능을 사용하여 다음과 같은 솔루션을 구현할 수 있습니다.

- 시각적 검색. 예를 들어 동일한 텍스트를 포함하는 이미지를 검색하고 표시합니다.
- 콘텐츠 분석 정보. 예를 들어, 추출된 비디오 프레임에서 인식되는 텍스트에서 발생하는 주제에 대한 분석 정보를 제공합니다. 애플리케이션은 뉴스, 스포츠 경기 점수, 운동 선수 번호 및 캡션과 같은 관련 콘텐츠에서 인식된 텍스트를 검색할 수 있습니다.
- 탐색. 예를 들어 레스토랑, 상점 또는 거리 표지의 이름을 인식하는 시각 장애인을 위한 음성 지원 모바일 앱을 개발합니다.
- 공공 안전과 운송 지원. 예를 들어 교통 카메라 이미지에서 자동차 번호판 번호를 감지합니다.
- 필터링. 예를 들어, 이미지에서 개인 식별 정보(PII)를 필터링합니다.

예를 들어 비디오에서 텍스트를 감지하는 다음과 같은 솔루션을 구현할 수 있습니다.

- 비디오에서 뉴스 화면에 게스트 이름과 같은 특정 텍스트 키워드가 있는 클립 검색
- 실수로 작성된 텍스트, 비속어 또는 스팸을 탐지하여 조직 표준을 준수하도록 콘텐츠 조절
- 콘텐츠 국제화를 위해 텍스트를 다른 언어로 된 텍스트로 대체하는 등 추가 처리를 위해 비디오 타임라인에서 모든 텍스트 오버레이 찾기
- 다른 그래픽을 적절하게 정렬할 수 있도록 텍스트 위치 찾기

JPEG 또는 PNG 형식의 이미지에서 텍스트를 감지하려면 DetectText 작업을 사용합니다. 비디오에서 텍스트를 비동기적으로 감지하려면 StartTextDetection 및 GetTextDetection 작업을 사용합니다. 이미지 및 비디오 텍스트 감지 작업은 고도의 스타일이 적용된 글꼴을 포함한 대부분의 글꼴을 지원합니다. 텍스트를 감지한 후 Amazon Rekognition은 감지된 단어와 텍스트 줄의 표현을 만들고, 그 사이의 관계를 보여 주며, 비디오의 이미지 또는 프레임에서 텍스트가 있는 위치를 알려줍니다.

`DetectText` 및 `GetTextDetection` 작업은 단어와 줄을 감지합니다. 단어는 공백으로 구분되지 않는 하나 이상의 스크립트 문자입니다. `DetectText`는 이미지에서 최대 100개의 단어를 감지할 수 있습니다. `GetTextDetection` 또한 비디오 프레임당 최대 100개의 단어를 감지할 수 있습니다.

단어는 공백으로 구분되지 않은 하나 이상의 문자입니다. Amazon Rekognition은 영어, 아랍어, 러시아어, 독일어, 프랑스어, 이탈리아어, 포르투갈어, 스페인어로 된 단어를 감지하도록 설계되었습니다.

줄은 동등하게 간격을 둔 단어로 구성된 문자열입니다. 한 줄이 반드시 완전한 문장은 아닙니다(마침표가 줄의 끝을 나타내지 않습니다). 예를 들어, Amazon Rekognition은 운전 면허증 번호를 한 줄로 감지합니다. 줄은 그 뒤에 정렬된 텍스트가 없거나 단어 조합의 길이에 비해 단어 사이에 큰 간격이 있을 때 끝납니다. 단어 사이의 간격에 따라 Amazon Rekognition은 같은 방향으로 정렬된 텍스트에서 줄 여러 개를 감지할 수 있습니다. 문장이 여러 줄에 걸쳐 있는 경우 작업에서 여러 줄을 반환합니다.

다음 이미지를 봐 주세요.

파란색 상자는 `DetectText` 작업이 반환하는 텍스트의 위치와 감지된 텍스트에 대한 정보를 나타냅니다. 이 예시에서 Amazon Rekognition은 "IT'S", "MONDAY", "but", "keep", "Smiling"을 단어로 감지합니다. Amazon Rekognition은 "IT'S", "MONDAY", "but keep", "Smiling"을 줄로 감지합니다. 텍스트가 감지되려면 텍스트가 가로 축의 +/- 90도 방향 내에 있어야 합니다.

예제는 이미지에서 텍스트 감지 섹션을 참조하세요.

주제

- 이미지에서 텍스트 감지
- 저장된 비디오에서 텍스트 감지

이미지에서 텍스트 감지

입력 이미지를 이미지 바이트 배열(base64 인코딩 이미지 바이트) 또는 Amazon S3 객체로 제공할 수 있습니다. 이 절차에서는 S3 버킷에 JPEG 또는 PNG 이미지를 업로드하고 파일 이름을 지정합니다.

이미지(API)에서 텍스트를 감지하려면

1. 아직 수행하지 않은 경우 다음 사전 조건을 완료하세요.

   a. `AmazonRekognitionFullAccess` 권한과 `AmazonS3ReadOnlyAccess` 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

   b. AWS Command Line Interface 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 텍스트를 포함하는 이미지를 S3 버킷에 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 예제를 사용하여 `DetectText` 작업을 호출합니다.

   Java

   다음 예제 코드는 이미지에서 감지된 줄과 단어를 표시합니다.

   `amzn-s3-demo-bucket` 및 `photo`의 값을 2단계에서 사용한 S3 버킷과 이미지의 이름으로 바꿉니다.

```
//Copyright 2018 Amazon.com, Inc. or its 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

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
/**
*  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

이 AWS CLI 명령은 detect-text CLI 작업에 대한 JSON 출력을 표시합니다.

amzn-s3-demo-bucket 및 Name의 값을 2단계에서 사용한 S3 버킷과 이미지의 이름으로 바꿉니다.

profile_name의 값을 개발자 프로필 이름으로 바꿉니다.

```
aws rekognition detect-text  --image "{"S3Object":{"Bucket":"amzn-s3-demo-
bucket","Name":"image-name"}}" --profile default
```

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예를 들어 다음을 참조하세요.

```
aws rekognition detect-text  --image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --profile default
```

Python

다음 예제 코드는 이미지에서 감지된 줄과 단어를 표시합니다.

amzn-s3-demo-bucket 및 photo의 값을 2단계에서 사용한 S3 버킷과 이미지의 이름으로 바꿉니다. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

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

import boto3

def detect_text(photo, bucket):

    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')

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

    textDetections = response['TextDetections']
    print('Detected text\n----------')
    for text in textDetections:
        print('Detected text:' + text['DetectedText'])
        print('Confidence: ' + "{:.2f}".format(text['Confidence']) + "%")
        print('Id: {}'.format(text['Id']))
        if 'ParentId' in text:
            print('Parent Id: {}'.format(text['ParentId']))
        print('Type:' + text['Type'])
        print()
    return len(textDetections)

def main():
    bucket = 'amzn-s3-demo-bucket'
    photo = 'photo-name'
    text_count = detect_text(photo, bucket)
    print("Text detected: " + str(text_count))

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

.NET

다음 예제 코드는 이미지에서 감지된 줄과 단어를 표시합니다.

amzn-s3-demo-bucket 및 photo의 값을 2단계에서 사용한 S3 버킷과 이미지의 이름으로 바꿉니다.

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

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

public class DetectText
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectTextRequest detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                }
            }
        };

        try
        {
            DetectTextResponse detectTextResponse =
 rekognitionClient.DetectText(detectTextRequest);
            Console.WriteLine("Detected lines and words for " + photo);
            foreach (TextDetection text in detectTextResponse.TextDetections)
```

```
            {
                Console.WriteLine("Detected: " + text.DetectedText);
                Console.WriteLine("Confidence: " + text.Confidence);
                Console.WriteLine("Id : " + text.Id);
                Console.WriteLine("Parent Id: " + text.ParentId);
                Console.WriteLine("Type: " + text.Type);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

Node.JS

다음 예제 코드는 이미지에서 감지된 줄과 단어를 표시합니다.

amzn-s3-demo-bucket 및 photo의 값을 2단계에서 사용한 S3 버킷과 이미지의 이름으로 바꿉니다. region의 값을 .aws 보안 인증 정보에서 확인할 수 있는 리전으로 바꾸세요. Rekognition 세션을 생성하는 라인에서 profile_name의 값을 개발자 프로필의 이름으로 대체합니다.

```
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 작업 요청

`DetectText` 작업에서 사용자는 입력 이미지를 base64로 인코딩된 바이트 배열 또는 Amazon S3 버킷에 저장된 이미지로 제공합니다. 다음 예제 JSON 요청은 Amazon S3 버킷에서 불러온 이미지를 표시합니다.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "inputtext.jpg"
        }
    }
}
```

필터

텍스트 영역, 크기 및 신뢰도 점수로 필터링하면 텍스트 감지 출력을 제어할 수 있는 추가적인 유연성이 제공됩니다. 관심 영역을 사용하면 텍스트 감지를 사용자와 관련된 영역으로 쉽게 제한할 수 있습니다. 프로필 사진의 오른쪽 상단이나 기계 이미지에서 부품 번호를 읽을 때 참조점을 기준으로 고정된 위치를 예로 들 수 있습니다. 단어 경계 상자 크기 필터는 정보 전달을 방해하거나 관련이 없는 작은 배

경 텍스트를 피하는 데 사용할 수 있습니다. 단어 신뢰도 필터를 사용하면 흐릿하거나 번져서 신뢰할 수 없는 결과를 제거할 수 있습니다.

필터 값에 대한 자세한 내용은 DetectTextFilters 섹션을 참조하세요.

다음 필터를 사용할 수 있습니다.

- MinConfidence - 단어 감지의 신뢰도를 설정합니다. 이 수준보다 감지 신뢰도가 낮은 단어는 결과에서 제외됩니다. 값은 0과 100 사이여야 합니다.
- MinBoundingBoxWidth - 단어 경계 상자의 최소 너비를 설정합니다. 경계 상자 너비가 이 값보다 작은 단어는 결과에서 제외됩니다. 이 값은 이미지 프레임 너비를 기준으로 합니다.
- MinBoundingBoxHeight - 단어 경계 상자의 최소 높이를 설정합니다. 경계 상자 높이가 이 값보다 작은 단어는 결과에서 제외됩니다. 이 값은 이미지 프레임 높이를 기준으로 합니다.
- RegionsOfInterest - 이미지 프레임의 특정 영역으로 감지를 제한합니다. 값은 프레임의 치수를 기준으로 합니다. 영역 내에 일부만 있는 텍스트의 경우 응답이 정의되지 않습니다.

DetectText 작업 응답

DetectText 작업은 이미지를 분석하고 TextDetections 배열을 반환합니다. 이때 각 요소(TextDetection)는 이미지에서 감지된 줄이나 단어를 나타냅니다. 각 요소마다 DetectText는 다음 정보를 반환합니다.

- 감지된 텍스트(DetectedText)
- 단어와 줄의 관계(Id 및 ParentId)
- 이미지에서 텍스트의 위치(Geometry)
- Amazon Rekognition이 감지한 텍스트와 경계 상자의 신뢰도(Confidence)
- 감지된 텍스트(Type)의 유형

감지된 텍스트

각 TextDetection 요소는 DetectedText 필드에서 인식된 텍스트(단어 또는 줄)를 포함합니다. 단어는 공백으로 구분되지 않은 하나 이상의 문자입니다. DetectText는 이미지에서 최대 100개의 단어를 감지할 수 있습니다. 반환된 텍스트에는 단어를 인식할 수 없도록 만드는 문자가 포함될 수 있습니다. 예를 들어 Cat 대신 C@t이 반환될 수 있습니다. TextDetection 요소가 텍스트 또는 단어로 구성된 줄을 나타내는지 확인하려면 Type 필드를 사용합니다.

각 TextDetection 요소에는 감지된 텍스트와 텍스트를 둘러싼 경계 상자의 정확도에 대한 Amazon Rekognition의 신뢰도를 나타내는 백분율 값이 포함됩니다.

단어와 줄의 관계

각 TextDetection 요소에는 식별자 필드 Id가 있습니다. Id는 줄에서 단어의 위치를 나타냅니다. 요소가 단어인 경우, 상위 식별자 ParentId는 단어가 감지되는 줄을 확인해 줍니다. 줄의 ParentId는 null입니다. 예를 들어 예제 이미지의 "but keep"이라는 줄에는 Id 및 ParentId 값이 있습니다.

텍스트	ID	상위 ID
but keep	3	
but	8	3
Keep	9	3

이미지에서 텍스트의 위치

이미지에서 인식된 텍스트의 위치를 확인하려면 DetectText가 반환하는 경계 상자(Geometry) 정보를 사용합니다. Geometry 객체에는 감지된 선과 단어에 대한 두 가지 유형의 경계 상자 정보가 있습니다.

- BoundingBox 객체의 축으로 정렬된 거친 직사각형 윤곽
- Point 배열에서 여러 개의 X와 Y 좌표로 구성된, 세분화된 다각형

테두리 상자와 다각형 좌표는 원본 이미지의 텍스트 위치를 나타냅니다. 좌표 값은 전체 이미지 크기의 비율입니다. 자세한 내용을 알아보려면 BoundingBox를 참조하세요.

DetectText 작업의 다음 JSON 응답은 다음 이미지에서 감지된 단어와 줄을 표시합니다.

```
{
 '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'}
```

저장된 비디오에서 텍스트 감지

Amazon Rekognition Video의 저장된 비디오 속 텍스트 감지는 비동기 작업입니다. 텍스트 감지를 시작하려면 StartTextDetection을 직접 호출하세요. Amazon Rekognition Video는 비디오 분석의 완료

상태를 Amazon SNS 주제에 게시합니다. 비디오 분석이 성공적으로 완료되면, GetTextDetection 직접 호출을 통해 분석 결과를 가져옵니다. 비디오 분석 시작 및 결과 가져오기에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오.

이 절차는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)의 코드를 확장하여 Amazon SQS 대기열을 사용해 비디오 분석 요청의 완료 상태를 가져옵니다.

Amazon S3 버킷에 저장된 비디오에서 텍스트를 감지하려면(SDK)

1. Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 단원의 단계를 따르십시오.
2. 1단계에서 `VideoDetect` 클래스에 다음 코드를 추가합니다.

   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);


}
```

main 함수에서 다음 줄을 바꿉니다.

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

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

다음으로 바꿉니다.

```
        StartTextDetection(amzn-s3-demo-bucket, video);

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

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
```

```
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

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

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

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

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

     String bucket = args[0];
     String video = args[1];
     String topicArn = args[2];
     String roleArn = args[3];

     Region region = Region.US_EAST_1;
```

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

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

    startTextLabels(rekClient, channel, bucket, video);
    GetTextResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

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

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

        StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
            .jobTag("DetectingLabels")
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
        startJobId = labelDetectionResponse.jobId();

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

```
        System.exit(1);
    }
}

public static void GetTextResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetTextDetectionResponse textDetectionResponse=null;
        boolean finished = false;
        String status;
        int yy=0 ;

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

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

            // Wait until the job succeeds.
            while (!finished) {
                textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                status = textDetectionResponse.jobStatusAsString();

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

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null.
            VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
```

```
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<TextDetectionResult> labels=
 textDetectionResponse.textDetections();
            for (TextDetectionResult detectedText: labels) {
                System.out.println("Confidence: " +
 detectedText.textDetection().confidence().toString());
                System.out.println("Id : " +
 detectedText.textDetection().id());
                System.out.println("Parent Id: " +
 detectedText.textDetection().parentId());
                System.out.println("Type: " +
 detectedText.textDetection().type());
                System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                System.out.println();
            }

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

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

Python

```
#Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

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

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

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

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

            print('Codec: ' + response['VideoMetadata']['Codec'])

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

            for textDetection in response['TextDetections']:
                text=textDetection['TextDetection']

                print("Timestamp: " + str(textDetection['Timestamp']))
                print("   Text Detected: " + text['DetectedText'])
                print("   Confidence: " +  str(text['Confidence']))
                print ("      Bounding box")
                print ("        Top: " + str(text['Geometry']['BoundingBox']
['Top']))
                print ("        Left: " + str(text['Geometry']['BoundingBox']
['Left']))
                print ("        Width: " +  str(text['Geometry']['BoundingBox']
['Width']))
                print ("        Height: " +  str(text['Geometry']['BoundingBox']
['Height']))
                print ("   Type: " + str(text['Type']) )
                print()

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

```
                finished = True
```

main 함수에서 다음 줄을 바꿉니다.

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

다음으로 바꿉니다.

```
    analyzer.StartTextDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetTextDetectionResults()
```

CLI

다음 AWS CLI 명령을 실행하여 비디오에서 텍스트 감지를 시작합니다.

```
 aws rekognition start-text-detection --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"video-name"}}"\
 --notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
 --region region-name --profile profile-name
```

다음 값을 업데이트합니다.

- amzn-s3-demo-bucket 및 video-name을 2단계에서 지정한 Amazon S3 버킷 이름과 파일 이름으로 변경합니다.
- region-name을 사용 중인 AWS 리전으로 변경합니다.
- profile-name의 값을 개발자 프로필 이름으로 바꿉니다.
- topic-ARN을 Amazon Rekognition Video 구성의 3단계에서 생성한 Amazon SNS 주제의 ARN으로 변경합니다.
- role-ARN을 Amazon Rekognition Video 구성의 7단계에서 생성한 IAM 서비스 역할의 ARN으로 변경합니다.

Windows 디바이스에서 CLI에 액세스하는 경우 작은따옴표 대신 큰따옴표를 사용하고 내부 큰따옴표는 백슬래시(즉 \)로 이스케이프 처리하여 발생할 수 있는 구문 분석 오류를 해결합니다. 예시를 보려면 다음을 참조하세요.

```
aws rekognition start-text-detection --video \
 "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"video-name\"}}" \
 --notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
 --region region-name --profile profile-name
```

진행 중인 코드 예제를 실행한 후 반환된 `jobID`를 복사하여 다음 `GetTextDetection` 명령에 제공하여 결과를 가져오고,`job-id-number`를 이전에 받은 `jobID`로 바꾸세요.

```
aws rekognition get-text-detection --job-id job-id-number --profile profile-name
```

> **Note**
>
> Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 이외에 비디오 예제를 이미 실행한 경우, 바꿀 코드가 다를 수 있습니다.

3. 코드를 실행합니다. 비디오에서 감지된 텍스트가 목록에 표시됩니다.

필터

필터는 `StartTextDetection`을 호출할 때 사용할 수 있는 선택적 요청 파라미터입니다. 텍스트 영역, 크기 및 신뢰도 점수로 필터링하면 텍스트 감지 출력을 제어할 수 있는 추가적인 유연성이 제공됩니다. 관심 영역을 사용하면 텍스트 감지를 사용자와 관련된 영역으로 쉽게 제한할 수 있습니다. 그래픽에서 하단 자막 영역 또는 축구 경기에서 득점판이 보이는 왼쪽 상단 코너를 예로 들 수 있습니다. 단어 경계 상자 크기 필터는 정보 전달을 방해하거나 관련이 없는 작은 배경 텍스트를 피하는 데 사용할 수 있습니다. 마지막으로, 단어 신뢰도 필터를 사용하면 흐릿하거나 번져서 신뢰할 수 없는 결과를 제거할 수 있습니다.

필터 값에 대한 자세한 내용은 `DetectTextFilters` 섹션을 참조하세요.

다음 필터를 사용할 수 있습니다.

- MinConfidence - 단어 감지의 신뢰도를 설정합니다. 이 수준보다 감지 신뢰도가 낮은 단어는 결과에서 제외됩니다. 값은 0과 100 사이여야 합니다.
- MinBoundingBoxWidth - 단어 경계 상자의 최소 너비를 설정합니다. 경계 상자 너비가 이 값보다 작은 단어는 결과에서 제외됩니다. 이 값은 비디오 프레임 너비를 기준으로 합니다.

- MinBoundingBoxHeight - 단어 경계 상자의 최소 높이를 설정합니다. 경계 상자 높이가 이 값보다 작은 단어는 결과에서 제외됩니다. 이 값은 비디오 프레임 높이를 기준으로 합니다.
- RegionsOfInterest - 프레임의 특정 영역으로 감지를 제한합니다. 값은 프레임 치수를 기준으로 합니다. 영역 내에 일부만 있는 객체의 경우 응답이 정의되지 않습니다.

GetTextDetection 응답

`GetTextDetection`은 비디오에서 감지된 텍스트에 대한 정보가 포함된 배열(`TextDetectionResults`)을 반환합니다. 배열 요소 TextDetection은 비디오에서 단어 또는 줄이 감지될 때마다 존재합니다. 배열 요소는 비디오 시작 후 시간별로(밀리초) 정렬됩니다.

다음은 `GetTextDetection`의 부분 JSON 응답입니다. 응답에서 다음에 유의하십시오.

- 텍스트 정보 - `TextDetectionResult` 배열 요소에는 감지된 텍스트(TextDetection)와 비디오에서 텍스트가 감지된 시간(`Timestamp`)에 관한 정보가 담겨 있습니다.
- 페이징 정보 - 이 예제는 텍스트 감지 정보의 페이지 하나를 보여줍니다. `GetTextDetection`의 `MaxResults` 입력 파라미터에 반환될 텍스트 요소의 수를 지정할 수 있습니다. `MaxResults`보다 많은 결과가 있거나 기본 최대값보다 많은 결과가 있는 경우 `GetTextDetection`은 결과의 다음 페이지를 가져올 때 사용되는 토큰(`NextToken`)을 반환합니다. 자세한 내용은 Amazon Rekognition Video 분석 결과 가져오기 단원을 참조하십시오.
- 비디오 정보 - 응답에는 `GetTextDetection`에서 반환된 정보의 각 페이지에 있는 비디오 형식(`VideoMetadata`)에 관한 정보가 포함되어 있습니다.

```
{
    "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"
}
```

저장된 비디오에서 비디오 세그먼트 감지

Amazon Rekognition Video는 블랙 프레임 및 엔드 크레딧과 같은 유용한 비디오 세그먼트를 식별하는 API를 제공합니다.

시청자들은 그 어느 때보다 많은 콘텐츠를 보고 있습니다. 특히 OTT(Over-The-Top) 및 VOD(온디맨드 비디오) 플랫폼을 통해 언제 어디서나 어떤 화면에서든 다양한 콘텐츠를 선택할 수 있습니다. 콘텐츠의 양이 급증하면서 미디어 기업들은 콘텐츠를 준비하고 관리하는 데 어려움을 겪고 있습니다. 콘텐츠 준비와 관리는 고품질의 시청 환경을 제공하고 더 나은 콘텐츠로 수익을 창출하는 데 매우 중요합니다. 오늘날 기업은 관련 교육을 이수한 대규모 인력 팀을 통해 다음과 같은 태스크를 수행하고 있습니다.

- 콘텐츠 내에서 오프닝 및 엔딩 크레딧 위치 찾기
- 광고를 삽입할 적절한 지점 선택(예: 무음 블랙 프레임 시퀀스)
- 더 나은 인덱싱을 위해 비디오를 작은 클립으로 분할

이러한 수작업 프로세스는 비용이 많이 들고 속도가 느릴 뿐만 아니라 매일 아카이브에서 생성, 라이선싱 및 검색되는 콘텐츠 양에 맞추어 규모를 조정할 수 없습니다.

Amazon Rekognition Video를 사용하면 기계 학습(ML)으로 구동되는 완전 관리형 전용 비디오 세그먼트 탐지 API를 사용하여 운영 미디어 분석 작업을 자동화할 수 있습니다. Amazon Rekognition Video 세그먼트 API를 사용하면 대량의 비디오를 손쉽게 분석하고 블랙 프레임이나 샷 전환 등의 마커를 감지할 수 있습니다. 각 감지에 대해 SMPTE(Society of Motion Picture and Television Engineers) 타임코드, 타임스탬프 및 프레임 번호가 생성됩니다. ML 경험은 필요하지 않습니다.

Amazon Rekognition Video는 Amazon Simple Storage Service(S3) 버킷에 저장된 동영상을 분석합니다. 반환되는 SMPTE 타임코드는 프레임 단위의 정밀성을 갖추고 있습니다. Amazon Rekognition Video는 감지된 비디오 세그먼트의 정확한 프레임 번호를 제공하고 다양한 비디오 프레임 속도 형식을 자동으로 처리합니다. Amazon Rekognition Video에서 정확한 프레임 정보를 갖춘 메타데이터를 사용하면 특정 태스크를 완전히 자동화할 수도 있고, 숙련된 인간 작업자의 검토 워크로드를 현저히 줄여 보다 창의적인 작업에 집중할 수 있게 지원합니다. 이를 통해 클라우드에서 콘텐츠 준비, 광고 삽입, 콘텐츠에 '몰아보기 마커' 추가 등의 태스크를 대규모로 수행할 수 있습니다.

요금에 대한 자세한 내용은 Amazon Rekognition 요금을 참조하세요.

Amazon Rekognition Video 세그먼트 감지는 두 가지 유형의 세그멘테이션 작업, 즉 기술적 큐 감지 및 샷 감지을 지원합니다.

주제

- 기술적 큐
- 샷 감지
- Amazon Rekognition Video 세그먼트 탐지 API 정보
- Amazon Rekognition Segment API 사용
- 예제: 저장된 비디오에서 세그먼트 감지

기술적 큐

기술적 큐는 비디오에서 블랙 프레임, 색상 막대, 오프닝 크레딧, 엔딩 크레딧, 스튜디오 로고, 기본 프로그램 콘텐츠를 식별합니다.

블랙 프레임

광고를 삽입하거나 장면 또는 오프닝 크레딧과 같은 프로그램 세그먼트의 끝을 표시하기 위한 신호로 사용되는 오디오가 없는 빈 블랙 프레임이 비디오에 포함된 경우가 많습니다. Amazon Rekognition Video를 사용하면 블랙 프레임 시퀀스를 감지하여 광고 삽입을 자동화하고, VOD 콘텐츠를 패키징하고, 다양한 프로그램 세그먼트나 장면을 구분할 수 있습니다. 페이드 아웃이나 음성 해설 등의 오디오가 있는 블랙 프레임은 콘텐츠로 간주되어 반환되지 않습니다.

Credits

Amazon Rekognition Video는 영화 또는 TV 프로그램에서 오프닝 및 엔딩 크레딧이 시작하고 끝나는 정확한 프레임을 자동으로 식별할 수 있습니다. 이 정보를 사용하여 VOD(비디오 온 디맨드) 애플리케이션의 비디오 내에 '다음 에피소드' 또는 '인트로 건너뛰기'와 같은 '몰아보기 마커' 또는 대화형 시청자 프롬프트를 생성할 수 있습니다. 비디오 내 프로그램 콘텐츠의 첫 번째 프레임과 마지막 프레임도 감지할 수 있습니다. Amazon Rekognition Video는 간단한 롤링 크레딧부터 콘텐츠와 함께 제공되는 더 까다로운 크레딧에 이르기까지 다양한 오프닝 및 엔딩 크레딧 스타일을 처리하도록 훈련되었습니다.

색상 막대

Amazon Rekognition Video를 사용하면 브로드캐스트 모니터, 프로그램 및 카메라에서 색상이 올바르게 보정되도록 특정 패턴으로 표시되는 색상 세트인 SMPTE 색상 막대를 표시하는 비디오 섹션을 감지할 수 있습니다. SMPTE 색상 막대에 대한 자세한 내용은 SMPTE 색상 막대를 참조하십시오. 이 메타데이터는 콘텐츠에서 색상 막대 세그먼트를 제거하여 VOD 애플리케이션용 콘텐츠를 준비하거나 색상 막대가 콘텐츠 대신 기본 신호로 계속 표시될 때 녹화에서 브로드캐스트 신호 손실과 같은 문제를 감지하는 데 유용합니다.

슬레이트

슬레이트는 일반적으로 시작 부분에 가까운 비디오 섹션으로, 에피소드, 스튜디오, 비디오 형식, 오디오 채널 등에 대한 텍스트 메타데이터를 포함합니다. Amazon Rekognition Video는 슬레이트의 시작과 끝을 식별할 수 있으므로 텍스트 메타데이터를 사용하거나 최종 시청을 위해 콘텐츠를 준비할 때 슬레이트를 제거하는 것이 쉽습니다.

스튜디오 로고

스튜디오 로고는 프로그램 제작에 참여한 프로덕션 스튜디오의 로고나 엠블럼을 보여주는 시퀀스입니다. Amazon Rekognition Video는 사용자가 검토를 통해 스튜디오를 식별할 수 있도록 이런 시퀀스를 감지합니다.

내용

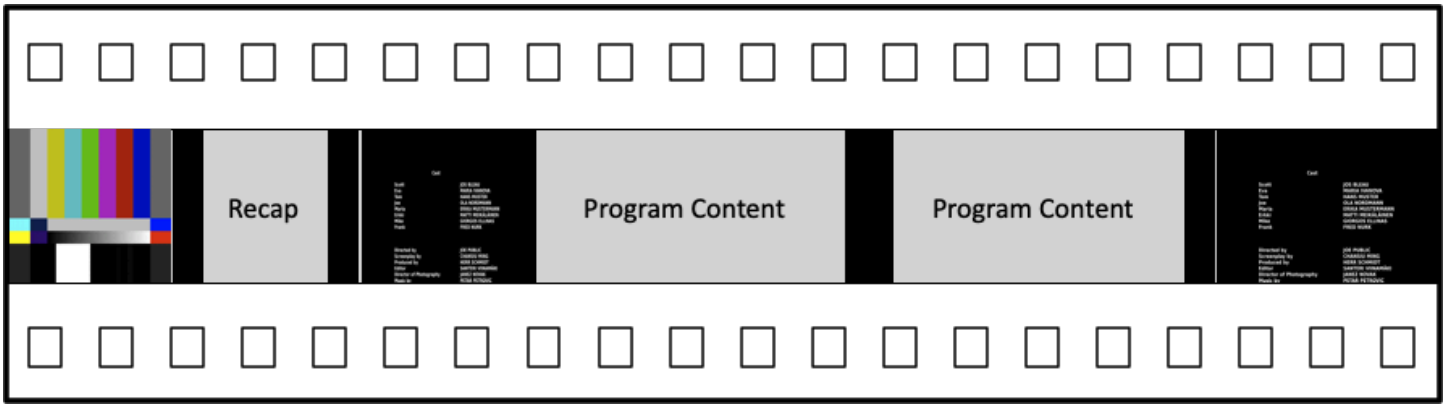
콘텐츠는 TV 프로그램 또는 영화에서 프로그램 또는 관련 요소가 포함된 부분입니다. 블랙 프레임, 크레딧, 색상 막대, 슬레이트, 스튜디오 로고는 콘텐츠로 간주되지 않습니다. Amazon Rekognition Video는 비디오에 있는 각 콘텐츠 세그먼트의 시작과 끝을 감지할 수 있으므로 사용자가 프로그램의 런타임 또는 특정 세그먼트를 찾을 수 있습니다.

콘텐츠 세그먼트는 다음이 포함되지만 이에 제한되지는 않습니다.

- 두 개의 광고 시간 사이에 들어가는 프로그램 장면
- 비디오 시작 부분에 나오는 이전 에피소드의 간단한 요약
- 크레딧 뒤에 나오는 보너스 콘텐츠
- '텍스트가 없는' 콘텐츠, 예를 들어 원래는 오버레이된 텍스트가 있었지만 다른 언어로 번역을 지원하기 위해 텍스트가 제거되어 있는 모든 프로그램 장면의 집합

Amazon Rekognition Video에서 모든 콘텐츠 세그먼트 탐지를 완료한 후에는 도메인 지식을 적용하거나 사람이 검토할 수 있도록 보내서 각 세그먼트를 추가로 분류할 수 있습니다. 예를 들어 사용자의 비디오가 항상 지난 에피소드 요약으로 시작한다면 첫 번째 콘텐츠 세그먼트를 요약으로 분류할 수 있습니다.

다음 다이어그램은 프로그램 또는 영화 타임라인의 기술적 큐 세그먼트를 보여 줍니다. 색상 막대와 오프닝 크레딧, 요약 및 메인 프로그램과 같은 콘텐츠 세그먼트, 동영상 전체의 검은색 프레임, 최종 크레딧이 있는 것을 확인하세요.



샷 감지

샷은 하나의 카메라로 연속 촬영한 서로 연결된 일련의 사진으로 시간과 공간에서 연속적인 동작을 나타냅니다. Amazon Rekognition Video를 사용하면 각 샷의 시작, 종료 및 재생 시간을 감지할 수 있을 뿐만 아니라 콘텐츠의 모든 샷을 셀 수 있습니다. 다음과 같은 태스크에 샷 메타데이터를 사용할 수 있습니다.

- 선택한 샷을 사용하여 프로모션 비디오 생성
- 샷 중간에 누군가가 말을 할 때와 같이 시청자의 경험을 방해하지 않는 위치에 광고 삽입
- 샷 간의 전환 콘텐츠를 방지하는 미리 보기 썸네일 세트 생성

샷 감지는 다른 카메라로의 확실한 전환이 있는 정확한 프레임에 표시됩니다. 한 카메라에서 다른 카메라로 부드럽게 전환되는 경우 Amazon Rekognition Video는 해당 전환을 생략합니다. 이를 통해 샷 시작 및 종료 시간에 실제 콘텐츠가 없는 섹션이 포함되지 않습니다.

다음 다이어그램은 필름 스트립의 샷 감지 세그먼트를 보여줍니다. 각 샷은 한 카메라 각도 또는 위치에서 다음 각도 또는 위치로 전환하는 것을 통해 식별됩니다.

Amazon Rekognition Video 세그먼트 탐지 API 정보

저장된 비디오를 세그먼트화하려면 비동기 StartSegmentDetection 및 GetSegmentDetection API 작업을 사용하여 세그먼테이션 작업을 시작하고 결과를 가져옵니다. 세그먼트 감지 기능은 Amazon S3 버킷에 저장된 비디오를 받아 JSON 출력을 반환합니다. StartSegmentdetection API 요청을 구성하여 기술적 큐만, 샷 전환만 또는 둘 다를 감지하도록 선택할 수 있습니다. 최소 예측 신뢰도에 대한 임계값을 설정하여 감지된 세그먼트를 필터링할 수도 있습니다. 자세한 내용은 Amazon Rekognition Segment API 사용 단원을 참조하십시오. 예제 코드는 예제: 저장된 비디오에서 세그먼트 감지 항목을 참조하세요.

Amazon Rekognition Segment API 사용

Amazon Rekognition Video의 저장된 비디오 속 얼굴 감지는 Amazon Rekognition Video 비동기 작업입니다. Amazon Rekognition Segment API는 단일 API 직접 호출에서 분석 유형(기술적 큐 또는 샷 감지)을 선택하는 복합 API입니다. 비동기 작업 호출에 대한 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오.

주제

- 세그먼트 분석 시작
- 세그먼트 분석 결과 가져오기

세그먼트 분석 시작

저장된 비디오에서 세그먼트 감지를 시작하려면 StartSegmentDetection을 직접 호출합니다. 입력 파라미터는 세그먼트 유형 선택 및 결과 필터링이 추가된다는 점을 제외하고 다른 Amazon Rekognition Video 작업과 동일합니다. 자세한 내용은 비디오 분석 시작 단원을 참조하십시오.

다음은 StartSegmentDetection에서 전달된 JSON의 예입니다. 이 요청에서는 기술적 큐와 샷 감지 세그먼트가 모두 감지되도록 지정하며, 각각에 서로 다른 최소 감지 신뢰도의 필터를 사용하고 있습니다(기술적 큐 세그먼트(90%), 샷 감지 세그먼트(80%)).

```
{
  "Video": {
    "S3Object": {
      "Bucket": "test_files",
      "Name": "test_file.mp4"
    }
    "SegmentTypes":["TECHNICAL_CUES", "SHOT"]
```

```
    "Filters": {
      "TechnicalCueFilter": {
         "MinSegmentConfidence": 90,
         "BlackFrame" : {
            "MaxPixelThreshold": 0.1,
            "MinCoveragePercentage": 95
         }
      },
      "ShotFilter" : {
          "MinSegmentConfidence": 60
      }
  }
}
```

세그먼트 유형 선택

`SegmentTypes` 배열 입력 파라미터를 사용하여 입력 비디오에서 기술적 큐 및/또는 샷 감지 세그먼트를 감지합니다.

- TECHNICAL_CUE - 비디오에서 감지된 기술적 큐(블랙 프레임, 색상 막대, 오프닝 크레딧, 엔딩 크레딧, 스튜디오 로고, 기본 프로그램 콘텐츠)의 시작, 종료 및 재생 시간에 대해 정확한 프레임 정보를 포함한 타임스탬프를 식별합니다. 예를 들어, 기술적 큐를 사용하여 엔딩 크레딧의 시작을 찾을 수 있습니다. 자세한 내용은 기술적 큐 단원을 참조하십시오.
- SHOT - 샷의 시작, 끝 및 재생 시간을 식별합니다. 예를 들어 샷 감지를 사용하여 비디오의 최종 편집을 위한 후보 샷을 식별할 수 있습니다. 자세한 내용은 샷 감지 단원을 참조하십시오.

분석 결과 필터링

`Filters`(StartSegmentDetectionFilters) 입력 파라미터를 사용하여 응답에서 반환되는 최소 감지 신뢰도를 지정할 수 있습니다. `Filters` 내에서 `ShotFilter`(StartShotDetectionFilter)를 사용하여 탐지된 샷을 필터링하세요. `TechnicalCueFilter`(StartTechnicalCueDetectionFilter)를 사용하여 기술적 큐를 필터링합니다.

예제 코드는 예제: 저장된 비디오에서 세그먼트 감지 항목을 참조하세요.

세그먼트 분석 결과 가져오기

Amazon Rekognition Video는 비디오 분석의 완료 상태를 Amazon Simple Notification Service 주제에 게시합니다. 비디오 분석이 성공적으로 완료되면 GetSegmentDetection을 직접 호출하여 비디오 분석 결과를 가져오세요.

다음은 예제 GetSegmentDetection 요청입니다. JobId는 StartSegmentDetection 호출에서 반환되는 작업 식별자입니다. 기타 입력 파라미터에 대한 자세한 내용은 Amazon Rekognition Video 분석 결과 가져오기 단원을 참조하십시오.

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
    "MaxResults": 10,
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw=="
}
```

GetSegmentDetection는 저장된 비디오에 대한 요청된 분석 결과와 일반 정보를 반환합니다.

일반 정보

GetSegmentDection는 다음과 같은 일반 정보를 반환합니다.

- 오디오 정보 - 응답에는 AudioMetadata 객체의 AudioMetadata 배열에 오디오 메타데이터가 포함됩니다. 여러 오디오 스트림이 있을 수 있습니다. 각 AudioMetadata 객체에는 단일 오디오 스트림에 대한 메타데이터가 포함됩니다. AudioMetadata 객체의 오디오 정보에는 오디오 코덱, 오디오 채널 수, 오디오 스트림 재생 시간 및 샘플 속도가 포함됩니다. 오디오 메타데이터는 GetSegmentDetection에서 반환한 정보의 각 페이지에 반환됩니다.
- 비디오 정보 - 현재 Amazon Rekognition Video는VideoMetadata 배열의 단일 VideoMetadata 객체를 반환합니다. 해당 객체에는 Amazon Rekognition Video가 분석하도록 선택한 입력 파일의 비디오 스트림에 대한 정보가 들어 있습니다. VideoMetadata 객체에는 비디오 코덱, 비디오 형식 및 기타 정보가 포함됩니다. 비디오 메타데이터는 GetSegmentDetection에서 반환한 정보의 각 페이지에 반환됩니다.
- 페이징 정보 - 이 예제는 세그먼트 정보의 페이지 하나를 보여 줍니다. GetSegmentDetection의 MaxResults 입력 파라미터에 반환될 요소의 수를 지정할 수 있습니다. MaxResults 보다 많은 결과가 존재할 경우 GetSegmentDetection은 결과의 다음 페이지를 가져올 때 사용되는 토큰(NextToken)을 반환합니다. 자세한 내용은 Amazon Rekognition Video 분석 결과 가져오기 단원을 참조하십시오.
- 요청 정보 - StartSegmentDetection 직접 호출에서 요청된 분석 유형이 SelectedSegmentTypes 필드에 반환됩니다.

Segments

비디오에서 감지된 기술적 큐와 샷 정보는 SegmentDetection 객체의 Segments 배열에 반환됩니다. 배열은 StartSegmentDetection의 SegmentTypes 입력 파라미터에 지정된 세그먼트 유형(TECHICAL_CUE 또는 SHOT)을 기준으로 정렬됩니다. 각 세그먼트 유형 내에서 배열은 타임스탬프 값으로 정렬됩니다. 각 SegmentDetection 객체에는 감지된 세그먼트 유형(기술적 큐 또는 샷 감지)에 대한 정보와 세그먼트 시작 시간, 종료 시간 및 재생 시간과 같은 일반 정보가 포함됩니다.

시간 정보는 세 가지 형식으로 반환됩니다.

- 밀리초

  비디오가 시작된 이후 경과한 시간(밀리초)입니다. DurationMillis, StartTimestampMillis 및 EndTimestampMillis 필드는 밀리초 형식입니다.
- 타임코드

  Amazon Rekognition Video 타임코드는 각 비디오 프레임에 고유한 타임코드 값이 있는 SMPTE 형식으로, hh:mm:ss:frame 형식을 갖습니다. 예를 들어 타임코드 값이 01:05:40:07이면 1시간, 5분, 40초, 7개 프레임을 의미합니다. Amazon Rekognition Video는 드롭 프레임 레이트 사용 사례를 지원합니다. 드롭 속도 타임코드 형식은 hh:mm:ss;frame입니다. DurationSMPTE, StartTimecodeSMPTE 및 EndTimecodeSMPTE 필드는 타임코드 형식입니다.
- 프레임 카운터

  각 비디오 세그먼트의 지속 시간도 프레임 수로 표시됩니다. StartFrameNumber 필드는 비디오 세그먼트의 시작 부분에 해당하는 프레임 번호를 제공하고 EndFrameNumber 필드는 비디오 세그먼트 끝의 프레임 번호를 제공합니다. DurationFrames 필드는 해당 비디오 세그먼트의 총 프레임 수를 알려줍니다. 이 값은 0으로 시작하는 프레임 인덱스를 사용하여 계산됩니다.

SegmentType 필드를 사용하여 Amazon Rekognition Video에서 반환하는 세그먼트의 유형을 확인할 수 있습니다.

- 기술적 큐 - TechnicalCueSegment 필드는 감지 신뢰도 및 기술적 큐 유형을 포함하는 TechnicalCueSegment 객체입니다. 기술적 큐의 유형은ColorBars, EndCredits, BlackFrames, OpeningCredits, StudioLogo, Slate, Content입니다.
- 샷 - ShotSegment 필드는 감지 신뢰도 및 비디오 내 샷 세그먼트의 식별자를 포함하는 ShotSegment 객체입니다.

다음 예제는 GetSegmentDetection의 JSON 응답입니다.

```
{
    "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
        }
    ]
}
```

예제 코드는 예제: 저장된 비디오에서 세그먼트 감지 항목을 참조하세요.

예제: 저장된 비디오에서 세그먼트 감지

다음 절차에서는 Amazon S3 버킷에 저장된 비디오에서 기술적 큐 세그먼트 및 샷 감지 세그먼트를 감지하는 방법을 보여 줍니다. 이 절차에서는 Amazon Rekognition Video가 감지 정확성에 대해 갖는 신뢰도를 기반으로 감지된 세그먼트를 필터링하는 방법도 보여 줍니다.

이 예제는 Amazon Simple Queue Service 대기열을 사용하여 비디오 분석 요청의 완료 상태를 가져오는 Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)의 코드를 확장합니다.

Amazon S3 버킷에 저장된 비디오에서 세그먼트를 감지하려면(SDK)

1. Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK)을 수행합니다.
2. 1단계에서 사용한 코드에 다음을 추가합니다.

   Java

   1. 다음과 같은 가져오기를 추가합니다.

```
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. 다음 코드를 VideoDetect 클래스에 추가합니다.

```
    //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. main 함수에서 다음 줄을 바꿉니다.

```
       StartLabelDetection(amzn-s3-demo-bucket, video);

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

다음으로 바꿉니다.

```
        StartSegmentDetection(amzn-s3-demo-bucket, video);

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

Java V2

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

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

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

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

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

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

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

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

    startTextLabels(rekClient, channel, bucket, video);
    GetTextResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_text.main]
public static void startTextLabels(RekognitionClient rekClient,
                                   NotificationChannel channel,
                                   String bucket,
```

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

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

        StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
            .jobTag("DetectingLabels")
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
        startJobId = labelDetectionResponse.jobId();

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

public static void GetTextResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetTextDetectionResponse textDetectionResponse=null;
        boolean finished = false;
        String status;
        int yy=0 ;

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

            GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                .jobId(startJobId)
```

```
                .nextToken(paginationToken)
                .maxResults(10)
                .build();

            // Wait until the job succeeds.
            while (!finished) {
                textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                status = textDetectionResponse.jobStatusAsString();

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

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null.
            VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<TextDetectionResult> labels=
textDetectionResponse.textDetections();
            for (TextDetectionResult detectedText: labels) {
                System.out.println("Confidence: " +
detectedText.textDetection().confidence().toString());
                System.out.println("Id : " +
detectedText.textDetection().id());
                System.out.println("Parent Id: " +
detectedText.textDetection().parentId());
                System.out.println("Type: " +
detectedText.textDetection().type());
                System.out.println("Text: " +
detectedText.textDetection().detectedText());
                System.out.println();
            }
```

```
        } while (textDetectionResponse !=null &&
 textDetectionResponse.nextToken() != null);

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

Python

1. 1단계에서 만든 클래스 VideoDetect에 다음 코드를 추가합니다.

```
# 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. main 함수에서 다음 줄을 바꿉니다.

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

```
        analyzer.GetLabelDetectionResults()
```

다음으로 바꿉니다.

```
    analyzer.StartSegmentDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetSegmentDetectionResults()
```

> **Note**
>
> Java 또는 Python으로 Amazon S3 버킷에 저장된 비디오 분석(SDK) 이외에 비디오 예제를 이미 실행한 경우, 바꿀 코드가 다를 수 있습니다.

3. 코드를 실행합니다. 입력 비디오에서 감지된 세그먼트에 대한 정보가 표시됩니다.

얼굴 생체 확인 감지

Amazon Rekognition Face Liveness를 사용하면 얼굴 인증을 받고 있는 사용자가 카메라 앞에 실제로 있는지 확인할 수 있습니다. 이는 카메라에 가해지거나 카메라를 우회하려는 스푸핑 공격을 탐지합니다. 사용자는 짧은 동영상 셀카를 찍어 자신의 존재를 확인하기 위한 일련의 프롬프트를 따라 얼굴 생체 확인 검사를 완료할 수 있습니다.

얼굴 생체 확인의 결과는 확률적 계산을 통해 결정되며, 검사 후 신뢰도 점수(0~100 사이)가 반환됩니다. 점수가 높을수록 검사를 받는 사람이 실제로 존재한다는 신뢰도가 높아집니다. Face Liveness는 또한 얼굴 비교 및 검색에 사용할 수 있는 참조 이미지라는 프레임을 반환합니다. 다른 확률 기반 시스템과 마찬가지로 Face Liveness는 완벽한 결과를 보장할 수 없습니다. 이를 다른 요인과 함께 사용하여 사용자의 개인 신원에 대한 위험 기반 결정을 내리세요.

Face Liveness는 다음과 같은 여러 구성 요소를 사용합니다.

- AWS FaceLivenessDetector 구성 요소가 포함된 Amplify SDK(React, Swift(iOS) 및 Android)
- AWS SDKs
- AWS 클라우드 APIs

Face Liveness 기능과 통합되도록 애플리케이션을 구성하면 다음과 같은 API 작업이 사용됩니다.

- CreateFaceLivenessSession - Face Liveness 감지 모델을 애플리케이션에서 사용할 수 있도록 Face Liveness 세션을 시작합니다. 생성된 세션의 SessionId를 반환합니다. 또한 FaceMovementChallenge 옵션을 사용할 수 있도록 ChallengePrefrence를 설정할 수 있습니다.
- StartFaceLivenessSession - AWS Amplify FaceLivenessDetector에서 호출합니다. 현재 세션의 관련 이벤트 및 속성에 대한 정보가 포함된 이벤트 스트림을 시작합니다.
- GetFaceLivenessSessionResults - Face Liveness 신뢰도 점수, 참조 이미지, 감사 이미지를 포함한 특정 Face Liveness 세션의 결과를 검색합니다.

AWS Amplify SDK를 사용하여 Face Liveness 기능을 웹 애플리케이션의 얼굴 기반 확인 워크플로와 통합합니다. 사용자가 애플리케이션을 통해 온보딩하거나 인증하면 Amplify SDK의 Face Liveness 검사 워크플로로 전송하세요. Amplify SDK는 사용자가 비디오 셀카를 찍는 동안 사용자 인터페이스 및 실시간 피드백을 처리합니다.

FaceMovementAndLightChallenge를 사용할 때 사용자의 얼굴이 디바이스에 표시된 타원형으로 이동하면 Amplify SDK가 화면에 일련의 컬러 조명을 표시합니다. 그 후 셀카 비디오를 클

라우드 API로 안전하게 스트리밍합니다. 또는 FaceMovementChallenge를 사용하면 사용자의 얼굴이 디바이스에 표시된 타원형으로 이동하지만 컬러 조명 시퀀스가 시작되지 않습니다. 'FaceMovementAndLightChallenge'는 정확도를 극대화하기 위한 최적의 설정으로 남아 있지만 'FaceMovementChallenge'를 사용하면 고객이 정확도보다 더 빠른 생체 확인을 우선시할 수 있습니다. 이러한 설정 중에서 선택할 때 고객은 예상 공격 유형, 원하는 거짓 수락 및 거짓 거부 비율을 포함한 사용 사례 요구 사항과 지리적 위치(예: IP 기반), 일회용 암호(OTP) 등과 같은 추가 검사를 고려해야 합니다. 고객은 특정 콘텐츠에 대한 다양한 신뢰도 점수 임계값으로 생체 확인 성능을 테스트한 후이 결정을 내려야 합니다. 또한 두 생체 확인 검사 유형 모두에서 애플리케이션 소유자는 비디오 스트림이 전송되는 디바이스를 보호하기 위한 제어를 구현해야 합니다. 분석이 완료되면 백엔드에서 다음을 받게 됩니다.

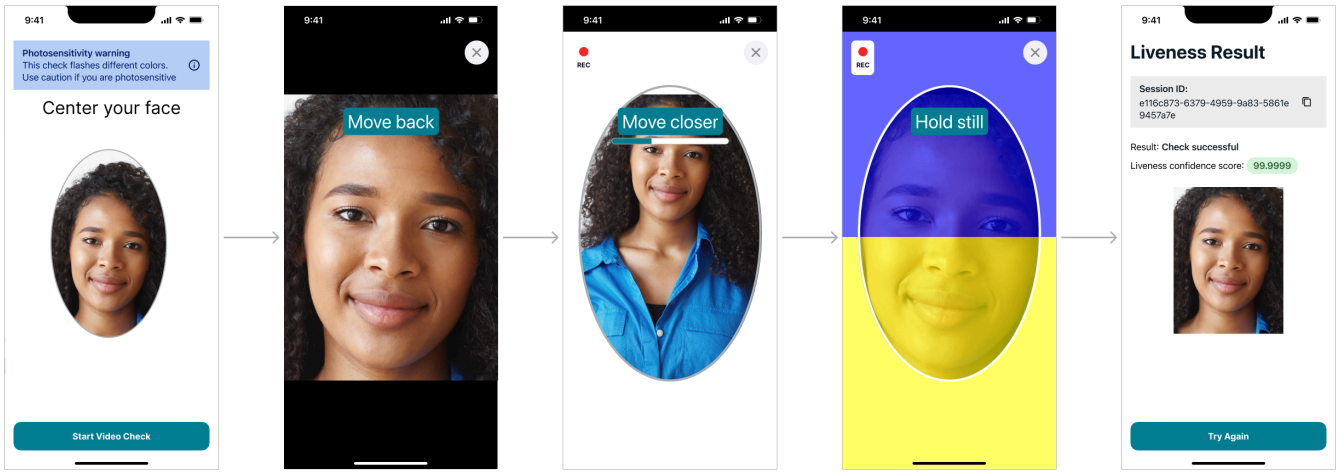
- Face Liveness 신뢰도 점수(0~100 사이)
- 얼굴 일치 또는 얼굴 검색에 사용할 수 있는 고품질 이미지(참조 이미지라고 함)
- 셀카 비디오에서 선택된 최대 4개의 이미지 집합(감사 이미지라고 함)

Face Liveness는 다양한 사용 사례에 활용할 수 있습니다. 예를 들어 Face Liveness를 얼굴 일치(CompareFaces 및 SearchFacesByImage와 함께 사용)와 함께 사용하여 신원 확인, 나이 기반 액세스 제한이 있는 플랫폼에서의 나이 추정, 봇 차단과 동시에 실제 인간 사용자를 감지할 수 있습니다.

Rekognition Face Liveness AI 서비스 카드에서 서비스의 사용 사례, 서비스에서 기계 학습(ML)을 사용하는 방법, 서비스의 책임 있는 설계 및 사용 시 주요 고려 사항에 대해 자세히 알아볼 수 있습니다.

Face Liveness 및 얼굴 일치 신뢰도 점수에 대한 임계값을 설정할 수 있습니다. 선택한 임계값은 여러분의 사용 사례를 반영해야 합니다. 그런 다음 점수가 임계값 초과 또는 미만임을 기준으로 사용자에게 신원 확인 승인 또는 거부 결과를 전송합니다. 거부된 경우 사용자에게 재시도를 요청하거나 다른 방법을 제시하세요.

다음 그림은 사용 지침부터 생체 확인 검사, 결과 반환에 이르는 사용자 흐름을 보여줍니다.



사용자 측 얼굴 생체 확인 요구 사항

Amazon Rekognition Face Liveness에는 다음과 같은 최소 사양이 필요합니다.

디바이스:

- 디바이스에는 전면 카메라가 있어야 합니다.
- 디바이스 디스플레이의 최소 화면 주사율: 60Hz
- 최소 디스플레이 또는 화면 크기: 4인치
- 탈옥되거나 루팅된 디바이스를 사용해서는 안 됩니다.

카메라 사양:

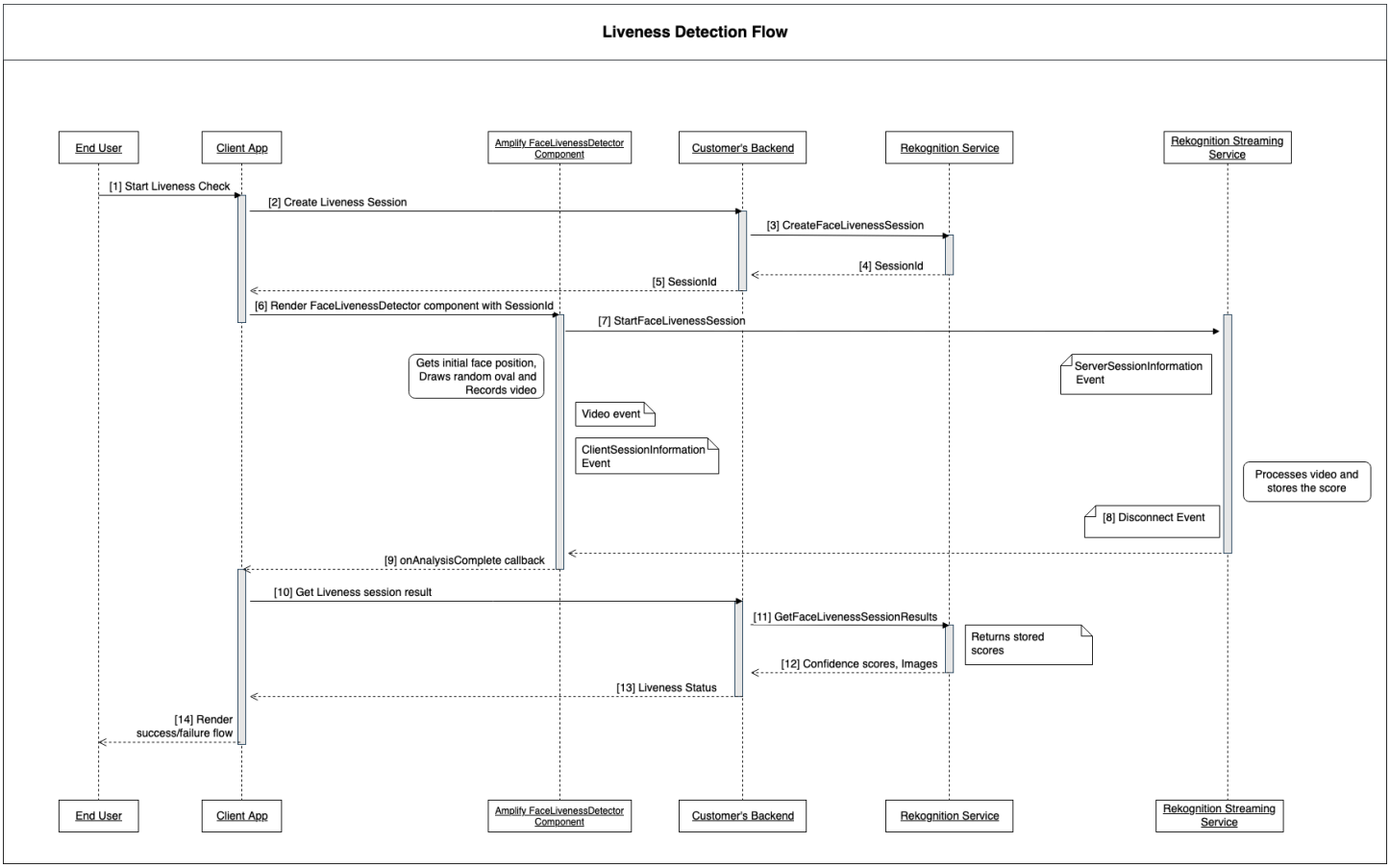
- 컬러 카메라: 전면 카메라는 색상을 기록할 수 있어야 합니다.
- 가상 카메라 또는 카메라 소프트웨어는 허용되지 않습니다.
- 최소 녹화 기능: 초당 15프레임.
- 최소 비디오 레코딩 해상도: 480x640px.
- 사용자가 Face Liveness 검사를 위해 데스크톱에서 웹캠을 사용하는 경우 Face Liveness 검사가 시작되는 동일한 화면 위에 웹캠을 장착하는 것이 중요합니다.

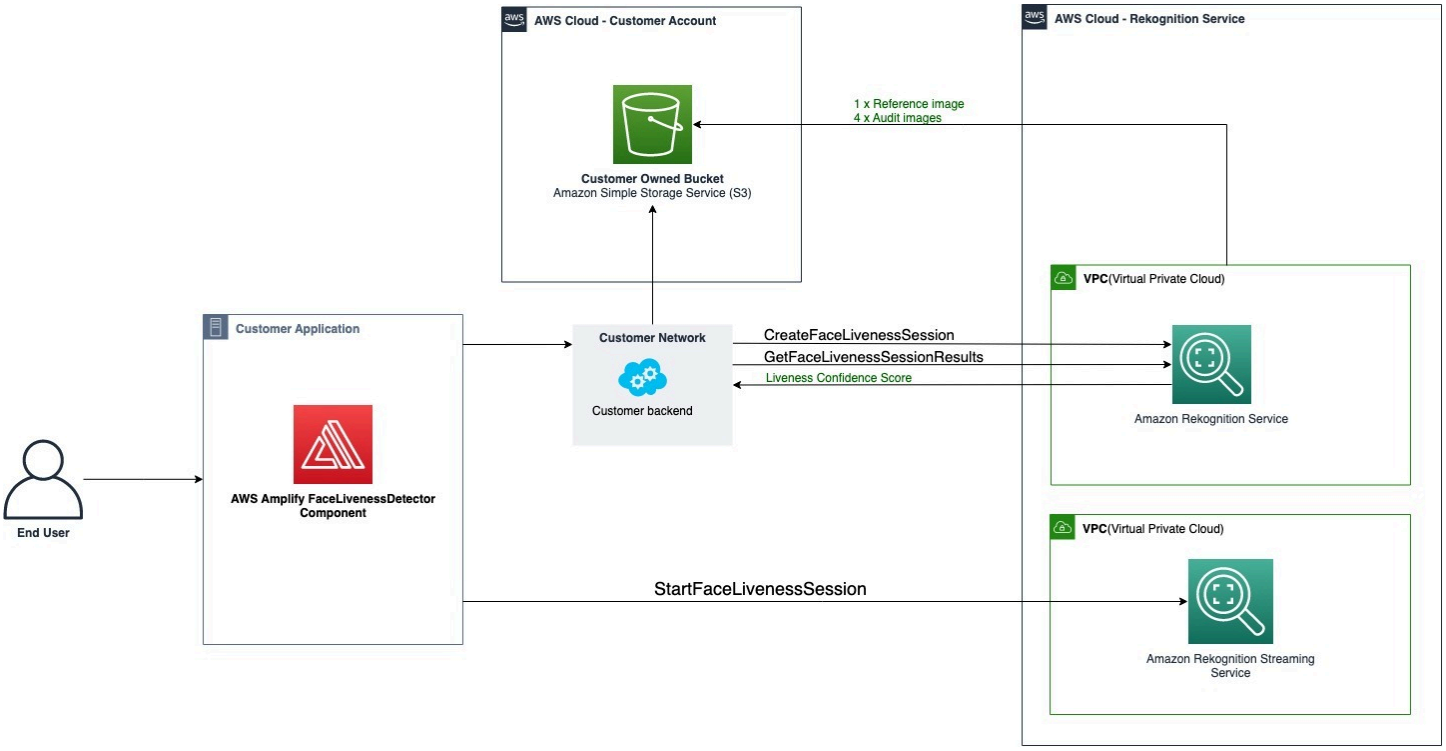
최소 대역폭 요구 사항: 100kbps

지원되는 브라우저: Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge 등 주요 브라우저의 최신 세 가지 버전. 브라우저 버전 지원에 대한 자세한 내용은 AWS Management Console을 사용할 수 있는 브라우저는 무엇입니까?를 참조하세요.

아키텍처 및 시퀀스 다이어그램

다음 다이어그램은 Amazon Rekognition Face Liveness의 아키텍처 및 작동 순서와 관련하여 이 기능이 작동하는 방식을 자세히 설명합니다.





Face Liveness 검사 프로세스에는 다음에 설명된 여러 단계가 포함됩니다.

1. 사용자가 클라이언트 앱에서 Face Liveness 검사를 시작합니다.
2. 클라이언트 앱은 고객의 백엔드를 직접 호출하고, 백엔드는 다시 Amazon Rekognition 서비스를 직접 호출합니다. 이 서비스는 Face Liveness 세션을 생성하고 고유한 SessionId를 반환합니다. 참고: SessionId가 전송된 후 3분 후에 만료되므로 아래 3~7단계를 완료하는 데 3분의 시간이 주어집니다. 모든 Face Liveness 검사에 새 sessionID를 사용해야 합니다. 지정된 sessionID를 후속 Face Liveness 검사에 사용하는 경우 검사에 실패합니다. 또한 SessionId는 전송된 후 3분 후에 만료되므로 세션과 연결된 모든 Liveness 데이터(예: sessionID, 참조 이미지, 감사 이미지 등)를 사용할 수 없습니다.
3. 클라이언트 앱은 획득한 SessionId와 적절한 콜백을 사용하여 FaceLivenessDetector Amplify 구성 요소를 렌더링합니다.
4. FaceLivenessDetector 구성 요소는 Amazon Rekognition 스트리밍 서비스에 대한 연결을 설정하고, 사용자 화면에 타원을 렌더링하고, 일련의 컬러 조명을 표시합니다. FaceLivenessDetector는 비디오를 녹화하여 Amazon Rekognition 스트리밍 서비스에 실시간으로 스트리밍합니다.
5. Amazon Rekognition 스트리밍 서비스는 비디오를 실시간으로 처리하고, 결과를 저장하고, 스트리밍이 완료되면 FaceLivenessDetector 구성 요소에 DisconnectEvent를 반환합니다.
6. FaceLivenessDetector 구성 요소는 `onAnalysisComplete` 콜백을 직접 호출하여 스트리밍이 완료되었으며 점수를 가져올 준비가 되었음을 클라이언트 앱에 알립니다.

7. 클라이언트 앱은 고객의 백엔드를 직접 호출하여 사용자의 생체 여부를 나타내는 부울 플래그를 가져옵니다. 고객 백엔드는 Amazon Rekognition 서비스에 신뢰도 점수, 참조 및 감사 이미지를 요청합니다. 고객 백엔드는 이러한 속성을 사용하여 사용자의 생체 여부를 판단하고 클라이언트 앱에 적절한 응답을 반환합니다.
8. 마지막으로 클라이언트 앱은 FaceLivenessDetector 구성 요소에 응답을 전달하고, FaceLivenessDetector 구성 요소는 성공 또는 실패 메시지를 적절하게 렌더링하여 흐름을 완료합니다.

사전 조건

Amazon Rekognition Face Liveness를 사용하기 위한 사전 요구 사항은 다음과 같습니다.

1. AWS 계정 설정
2. Face AWS SDKs 설정
3. AWS Amplify 리소스 설정

1단계: AWS 계정 설정

아직 AWS 계정이 없는 경우에 표시된 단계를 완료AWS 계정 및 사용자 생성하여 계정을 생성합니다.

2단계: Face AWS SDKs 설정

아직 설치하지 않았다면 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

AWS SDK 호출을 인증하는 방법에는 여러 가지가 있습니다. 이 가이드의 예제에서는 AWS CLI 명령 및 AWS SDK API 작업을 호출하기 위해 기본 자격 증명 프로파일을 사용하고 있다고 가정합니다.

사용자 계정에 선택한 AWS SDK에 대한 액세스 권한을 부여하는 방법에 대한 자세한 내용은 프로그래밍 방식 액세스 권한 부여 페이지를 참조하세요. 이 페이지에서는 로컬 컴퓨터에서 프로파일을 사용하는 방법과 AWS 환경에서 샘플 코드를 실행하는 방법도 설명합니다.

Face Liveness 작업을 직접 호출하는 사용자에게 작업을 직접 호출할 수 있는 올바른 권한(예: `AmazonRekognitionFullAccess` 및 `AmazonS3FullAccess` 권한)이 있는지 확인하세요.

3단계: AWS Amplify 리소스 설정

Amazon Rekognition Face Liveness를 앱에 통합하려면 FaceLivenessDetector Amplify 구성 요소를 사용할 수 있도록 AWS Amplify SDK를 설정해야 합니다.

아직 설정하지 않았다면, AWS CLI 시작에서 AWS Command Line Interface(AWS CLI)를 설정하는 지침을 따르세요. CLI를 설치한 후 Amplify UI 문서 사이트에 표시된 인증 구성 단계 단계를 완료하여 AWS Amplify 리소스를 설정합니다.

Face Liveness 감지 모범 사례

Amazon Rekognition Face Liveness를 사용할 때 몇 가지 모범 사례를 따르는 것이 좋습니다. Face Liveness 모범 사례는 Face Liveness 검사를 수행해야 하는 위치, 감사 이미지 사용, 신뢰도 점수 임계값 선택에 대한 가이드라인을 포함합니다.

모범 사례의 전체 목록은 Face Liveness 사용에 대한 권장 사항 섹션을 참조하세요.

Amazon Rekognition Face Liveness API 프로그래밍

Amazon Rekognition Face Liveness API를 사용하려면 다음 단계를 수행하는 백엔드를 생성해야 합니다.

1. CreateFaceLivenessSession을 직접 호출하여 Face Liveness 세션을 시작하세요. `CreateFaceLivenessSession` 작업이 완료되면 사용자에게 비디오 셀카를 제출하라는 메시지가 표시됩니다. 그러면 AWS Amplify의 FaceLivenessDetector 구성 요소가 StartFaceLivenessSession을 직접 호출하여 생체 확인 감지를 수행합니다.
2. GetFaceLivenessSessionResults를 직접 호출하여 Face Liveness 세션에 연결된 탐지 결과를 반환합니다.
3. Amplify Liveness 설명서의 단계에 따라 FaceLivenessDetector 구성 요소를 사용할 수 있도록 React 애플리케이션을 구성하세요.

Face Liveness를 사용하기 전에 AWS 계정을 생성하고, AWS CLI 및 AWS SDK를 설정하고, AWS Amplify를 설정했는지 확인하세요. 또한 백엔드 API의 IAM 정책에 `GetFaceLivenessSessionResults` 및 `CreateFaceLivenessSession`을 포함하는 권한이 있는지 확인해야 합니다. 자세한 내용은 사전 조건 섹션을 참조하세요.

1단계: CreateFaceLivenessSession

CreateFaceLivenessSession API 작업은 Face Liveness 세션을 생성하고 고유한 `SessionId`를 반환합니다.

이 작업에 대한 입력의 일부로 Amazon S3 버킷 위치를 지정할 수도 있습니다. 이를 통해 Face Liveness 세션 중에 생성된 참조 이미지와 감사 이미지를 저장할 수 있습니다. Amazon S3 버킷은 호출자의 AWS 계정 및 Face Liveness 엔드포인트와 동일한 리전에 있어야 합니다. 또한 S3 객체 키는 Face Liveness 시스템에 의해 생성됩니다.

0에서 4 사이의 숫자인 `AuditImagesLimit`을 제공할 수도 있습니다. 이는 기본적으로 0으로 설정됩니다. 반환되는 이미지 수는 셀카 동영상의 재생 시간을 기준으로 한 최선의 결과입니다.

마지막으로 세션의 기본값을 지정할 챌린지의 유형 및 버전을 포함하여 ChallengePreference를 제공할 수 있습니다. 기본적으로 FaceLivenessMovementAndLightChallenge를 사용하지만 FaceMovementChallenge로 설정할 수 있습니다.

요청 예제

```
{
   "ClientRequestToken": "string",
   "KmsKeyId": "string",
   "Settings": {
      "AuditImagesLimit": number,
      "ChallengePreferences": [
         {
            "Type": "string",
            "Versions": {
               "Maximum": "string",
               "Minimum": "string"
            }
         }
      ],
      "OutputConfig": {
         "S3Bucket": "string",
         "S3KeyPrefix": "string"
      }
   }
}
```

응답 예제

```
{
    {"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
}
```

2단계: StartFaceLivenessSession

CreateFaceLivenessSession API 작업이 완료되면 AWS Amplify 구성 요소가 StartFaceLivenessSession API 작업을 수행합니다. 사용자에게 비디오 셀카를 찍으라는 메시지가 표시됩니다. 검사에 통과하려면 사용자는 조명을 잘 유지하면서 화면의 타원형 안에 얼굴을 위치시켜야 합니다. 자세한 내용은 Face Liveness 사용에 대한 권장 사항 단원을 참조하십시오.

이 API 작업을 수행하려면 Face Liveness 세션 중에 캡처한 비디오, CreateFaceLivenessSession API 작업에서 얻은 sessionId 및 `onAnalysisComplete` 콜백이 필요합니다. 이 콜백을 사용해서 신뢰도 점수, 참조 및 감사 이미지를 반환하는 GetFaceLivenessSessionResults API 작업을 직접 호출하도록 백엔드에 신호를 보낼 수 있습니다.

이 단계는 클라이언트 애플리케이션의 AWS Amplify FaceLivenessDetector 구성 요소에 의해 수행된다는 점에 유의하세요. `StartFaceLivenessSession`을 직접 호출하기 위한 추가 설정이 필요 없습니다.

3단계: GetFaceLivenessSessionResults

GetFaceLivenessSessionResults API 작업은 특정 Face Liveness 세션의 결과를 가져옵니다. sessionId 입력이 필요하고 그에 해당하는 Face Liveness 신뢰도 점수를 반환합니다. 또한 얼굴 경계 상자가 포함된 참조 이미지와 역시 얼굴 경계 상자가 포함된 감사 이미지도 제공합니다. Face Liveness 신뢰도 점수 범위는 0~100입니다.

요청 예제

```
{"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
```

응답 예제

```
{
    "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"
}
```

4단계: 결과에 응답

Face Liveness 세션이 끝나면 검사의 신뢰도 점수를 지정된 임계값과 비교하세요. 점수가 임계값보다 높으면 사용자는 다음 화면 또는 작업으로 이동할 수 있습니다. 검사에 실패하면 사용자에게 알림이 표시되고 다시 시도하라는 메시지가 표시됩니다.

Face Liveness API 직접 호출

AWS Python AWS SDK Boto3 또는 Java용 AWS SDK와 같이 지원되는 SDK를 사용하여 Amazon Rekognition Face Liveness를 테스트할 수 있습니다. Boto3 선택한 SDK로 `CreateFaceLivenessSession` 및 `GetFaceLivenessSessionResults` API를 직접 호출할 수 있습니다. 다음 섹션에서는 Python 및 Java SDK를 사용하여 이러한 API를 직접 호출하는 방법을 보여줍니다.

Face Liveness API를 직접 호출하려면

- 아직 AmazonRekognitionFullAccess 권한이 있는 사용자를 생성하거나 업데이트하지 않았다면 이제 하세요. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성을 참조하세요.
- 아직 AWS CLI 및 AWS SDK를 설치하고 구성하지 않았다면 이제 하세요. 자세한 내용은 2단계: AWS CLI 및 AWS SDK 설정을 참조하세요.

Python

다음 코드 조각은 Python 애플리케이션에서 이러한 API를 직접 호출하는 방법을 보여줍니다. 참고로 이 예제를 실행하려면 Boto3 SDK 버전 1.26.110 이상을 사용해야 하지만 가장 최신 버전의 SDK를 사용하는 것이 좋습니다.

```
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

다음 코드 조각은 Java 애플리케이션에서 이러한 API를 직접 호출하는 방법을 보여줍니다.

```
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

다음 코드 조각은 AWS Java V2 SDK를 사용하여 Face Liveness APIs를 호출하는 방법을 보여줍니다.

```
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

다음 코드 조각은 AWS Node.Js SDK를 사용하여 Face Liveness APIs를 호출하는 방법을 보여줍니다.

```
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)

다음 코드 조각은 Javascript v3용 AWS Node.Js SDK를 사용하여 Face Liveness APIs를 호출하는 방법을 보여줍니다.

```
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
// };
```

애플리케이션 구성 및 사용자 지정

애플리케이션 구성

Face Liveness 애플리케이션은 모바일 디바이스 또는 데스크톱 웹 브라우저에서 작동할 수 있습니다. 선택한 솔루션과 통합되도록 Face Liveness 구성 요소를 구성하는 것이 좋습니다. 또한 애플리케이션에 디바이스의 카메라를 사용할 권한이 있는지 확인해야 합니다. Amplify Liveness 설명서에서는 다음을 수행하는 방법에 대한 자세한 지침을 제공합니다.

- AWS Amplify의 설치 및 구성
- FaceLivenessDetector 구성 요소 가져오기 및 렌더링
- 콜백 듣기
- Amplify 예제 오류 메시지 렌더링

애플리케이션 사용자 지정

AWS Amplify를 사용하여 생체 확인 애플리케이션의 특정 구성 요소를 사용자 지정할 수 있습니다.

번역에 대한 자세한 내용은 Amplify Authenticator 설명서를 참조하세요.

Amplify 구성 요소 및 테마를 사용자 지정하는 방법에 대한 자세한 내용은 테마 지정에 관한 Amplify 설명서를 참조하세요.

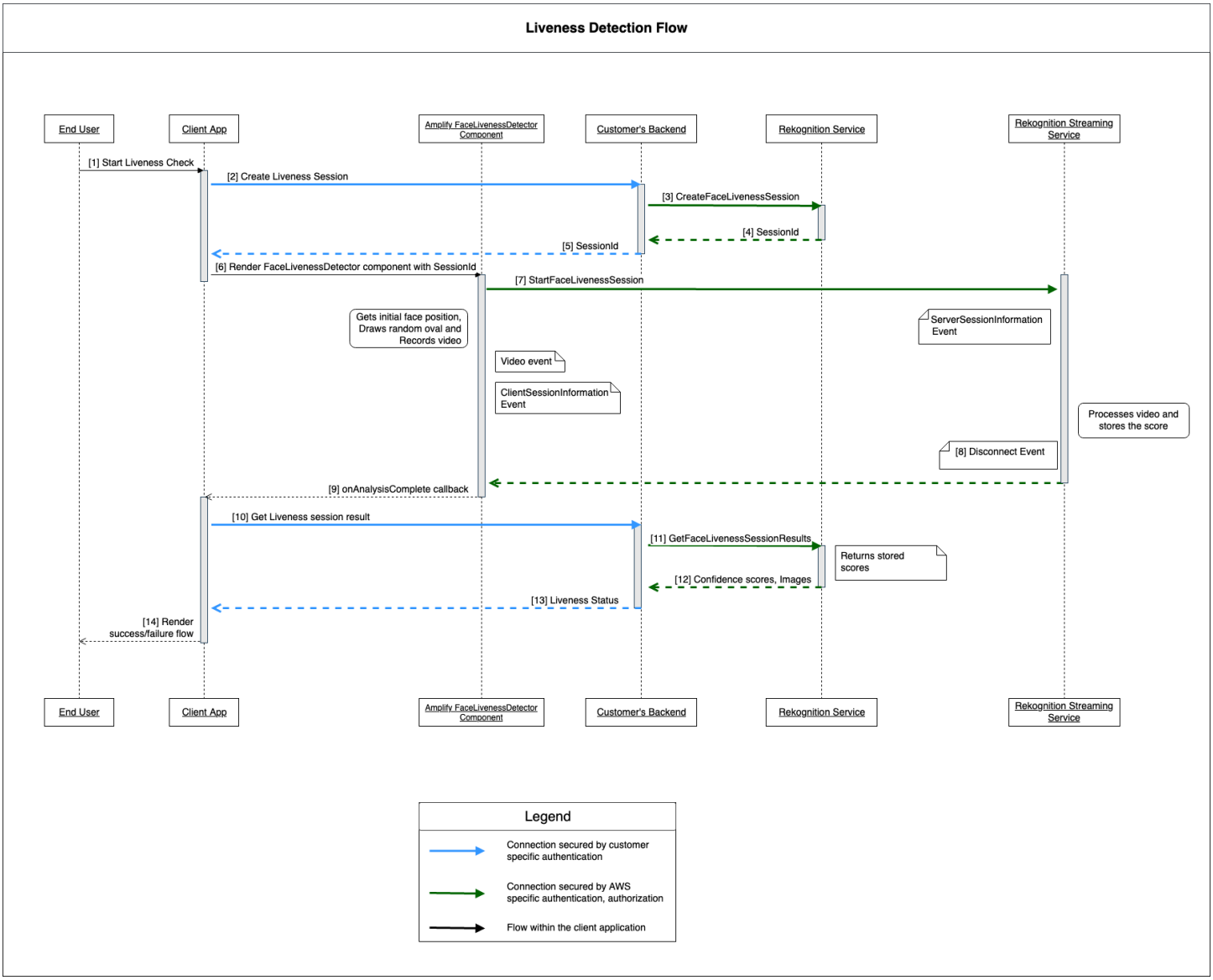
Face Liveness 공동 책임 모델

보안과 규정 준수는 AWS와 고객의 공동 책임입니다. AWS 공동 책임 모델에 대한 자세한 내용은 여기를 참조하세요.

1. (클라이언트 애플리케이션 또는 고객 백엔드를 통한) 모든 AWS 서비스 직접 호출은 AWS Auth(AWSAuthentication)를 통해 인증 및 승인됩니다. 이를 보장하는 것은 Face Liveness 서비스 소유자의 책임입니다.
2. (클라이언트 애플리케이션에서) 고객 백엔드로 보내는 모든 직접 호출은 고객을 통해 인증 및 권한이 부여됩니다. 이 책임은 고객에게 있습니다. 고객은 클라이언트 애플리케이션에서 오는 직접 호출 인증을 확인하고 어떤 방식으로든 조작되지 않도록 해야 합니다.
3. 고객 백엔드는 Face Liveness 과제를 수행하는 최종 사용자를 식별해야 합니다. 최종 사용자를 Face Liveness 세션에 연결하는 것은 고객의 책임입니다. Face Liveness 서비스는 최종 사용자간의 차이를 구분하지 않습니다. (고객이 처리하는) 직접 호출 AWS ID만 식별할 수 있습니다.
4. AWS는 고객이 사용 사례 요구 사항 및 보안 태세에 맞는 Face Liveness 외에도 위치 지리적 위치(예: IP 기반), 일회용 패스 코드(OTP) 등과 같은 추가 검증 검사를 적용할 것을 권장합니다.

'FaceMovementAndLightChallenge' 설정은 사용자가 얼굴을 화면 쪽으로 움직이고 일련의 깜박이는 불빛이 켜질 때까지 가만히 있어야 하므로 Rekognition Liveness에 가장 높은 정확도를 제공합니다. 고객은 이 기본 설정을 사용하는 것이 좋습니다. 또는 고객은 ‘FaceMovementChallenge’ 설정을 활성화할 수 있습니다. 이 설정은 깜빡이는 불빛을 제거하여 확인 시간을 몇 초 단축할 수 있습니다. 'FaceMovementAndLightChallenge'는 정확도를 극대화하기 위한 최적의 설정으로 남아 있지만 'FaceMovementChallenge'를 사용하면 고객이 더 빠른 생체 확인을 우선시할 수 있습니다. 이러한 설정 중에서 선택할 때 고객은 예상 공격 유형, 원하는 거짓 수락 및 거짓 거부 비율을 포함한 사용 사례 요구 사항을 고려하고 지리적 위치(예: IP 기반), 일회용 암호(OTP) 등과 같은 추가 검사를 구현해야 합니다. 고객은 사용 사례에 따라 다양한 신뢰도 점수 임계값으로 생체 확인 성능을 테스트한 후이 결정을 내려야 합니다. 고객은 비디오가 전송되는 디바이스를 보호하기 위한 제어 수단을 구현할 책임이 있습니다.

다음 흐름도는 어떤 직접 호출이 AWS 서비스 또는 고객에 의해 인증되는지를 보여 줍니다.



Amazon Rekognition Face Liveness 서비스에 대한 모든 직접 호출은(AWS 서명 메커니즘을 사용하는) AWS Auth에 의해 보호됩니다. 이는 다음과 같은 직접 호출을 포함합니다.

- [3] CreateFaceLivenessSession API 직접 호출(고객 백엔드에서 발신)
- [7] StartFaceLivenessSession API 직접 호출(클라이언트 애플리케이션에서 발신)
- [11] GetFaceLivenessSessionResults API 직접 호출(고객 백엔드에서 발신)

고객의 백엔드로 전송되는 모든 직접 호출에는 인증 및 권한 부여 메커니즘이 있어야 합니다. 고객은 사용된 서드 파티 코드 및 라이브러리 등이 적극적으로 유지 관리 및 개발되고 있는지 확인해야 합니다. 또한 고객은 올바른 최종 사용자가 올바른 Face Liveness 세션으로 직접 호출을 보내고 있는지 확인해야 합니다. 고객은 다음 흐름을 인증하고 권한을 부여하여야 합니다.

- [2] Face Liveness 세션 생성(클라이언트 애플리케이션에서)
- [10] Face Liveness 세션 결과 가져오기(클라이언트 애플리케이션에서)

고객은 STRIDE 보안 모델을 따름으로써 API 직접 호출을 확실히 보호할 수 있습니다.

유형	설명	보안 제어
Spoofing	Threat action aimed at accessing and use of another user's credentials, such as username and password.	Authentication
Tampering	Threat action intending to maliciously change or modify persistent data. Examples include records in a database, and the alteration of data in transit between two computers over an open network, such as the internet.	Integrity
Repudiation	Threat action aimed at performing prohibited operations in a system that lacks the ability to trace the operations.	Non-Repudiation
Information disclosure	Threat action intending to read a file that one was not granted access to, or to read data in transit.	Confidentiality
Denial of service	Threat action attempting to deny access to valid users, such as by making a web server temporarily unavailable or unusable.	Availability

Elevation of privilege	Threat action intending to gain privileged access to resources in order to gain unauthorized access to information or to compromise a system.	Authorization

AWS는 다음과 같은 방법으로 연결을 보호합니다.

1. 요청의 서명을 추정한 다음 서비스 측에서 서명을 확인합니다. 요청은 이 서명을 사용하여 인증됩니다.
2. AWS 고객은 적절한 IAM 역할을 설정하여 특정 작업에 권한을 부여해야 합니다. 이러한 IAM 역할은 AWS 서비스에 직접 호출을 보내는 데 필요합니다.
3. AWS 서비스에는 HTTPS 요청만 허용됩니다. 요청은 TLS를 사용하여 개방형 네트워크에서 암호화됩니다. 이렇게 하면 요청의 기밀성이 보호되고 요청 무결성이 유지됩니다.
4. AWS 서비스는 고객이 보낸 직접 호출을 식별하기에 충분한 양의 데이터를 기록합니다. 이렇게 하면 부인 공격을 방지할 수 있습니다.
5. AWS 서비스는 충분한 가용성을 유지할 책임을 집니다.

고객은 다음과 같은 방법으로 서비스 및 API 직접 호출을 보호할 책임이 있습니다.

1. 고객은 적절한 인증 메커니즘을 따라야만 합니다. 요청을 인증하는 데 사용할 수 있는 다양한 인증 메커니즘이 있습니다. 고객은 다이제스트 기반 인증, OAuth, OpenID 및 기타 메커니즘을 탐색해 볼 수 있습니다.
2. 고객은 서비스가 서비스 API 직접 호출을 위한 적절한 암호화 채널(예: TLS/HTTPS)을 지원하도록 보장해야 합니다.
3. 고객은 API 직접 호출과 호출자의 고유 식별에 필요한 데이터를 기록해야 합니다. 정의된 파라미터와 호출 시간을 사용하여 고객의 API를 직접 호출하는 클라이언트를 식별할 수 있어야 합니다.
4. 고객은 시스템이 사용 가능한 상태이며 DDoS 공격에서 보호받도록 해야 합니다. 다음은 DDoS 공격에 대한 방어 기법의 몇 가지 예입니다.

고객은 애플리케이션을 최신 상태로 유지할 책임이 있습니다. 자세한 내용은 Face Liveness 업데이트 가이드라인 섹션을 참조하세요.

Face Liveness 업데이트 가이드라인

AWS 는 정기적으로 Face AWS SDKs(고객 백엔드에 사용됨) 및 AWS Amplify SDKs의 FaceLivenessDetector 구성 요소(클라이언트 애플리케이션에 사용됨)를 업데이트하여 새로운 기능, 업데이트된 APIs, 향상된 보안, 버그 수정, 사용성 개선 등을 제공합니다. 해당 기능이 최적으로 작동할 수 있도록 SDK를 최신 상태로 유지하는 것이 좋습니다. 이전 버전의 SDK를 계속 사용하는 경우 유지 관리 및 보안상의 이유로 요청이 차단될 수 있습니다.

Face Liveness를 사용하려면 AWS Amplify SDKs(React, iOS, Android)에 포함된 FaceLivenessDetector 구성 요소를 사용해야 합니다.

버전 관리 및 기간 범위

Face Liveness 기능의 다음 주요 구성 요소는 버전 관리되고 있습니다. 시맨틱 버전 관리 형식을 따르고 있습니다. 예를 들어, X.Y.Z의 버전 형식에서 X는 메이저 버전, Y는 마이너 버전, Z는 패치 버전에 해당합니다.

- Face Liveness 사용자 챌린지(예: FaceMovementAndLightChallenge 챌린지)는 StartFaceLivenessSession API의 일부입니다.
- AWS Amplify SDKs 통해 제공되는 FaceLivenessDetector 구성 요소는 클라이언트 애플리케이션에서 사용됩니다.

메이저 버전: 메이저 버전 업데이트는 중대 보안, 최신 API, 눈에 띄는 사용성 업데이트를 위한 것입니다. Face Liveness 기능을 계속 사용하려면 애플리케이션과 고객 백엔드를 최대한 빠르게 업데이트해야 합니다. 새 메이저 버전이 출시되면 출시일로부터 120일 동안 지난 메이저 버전을 지원합니다. 120일이 지나면 지난 메이저 버전에서 오는 요청이 차단될 수 있습니다.

마이너 버전: 마이너 버전 업데이트는 중요한 보안과 사용 편의성 기능 및 개선을 위한 것입니다. 마이너 버전 업데이트를 적용하시기를 권고합니다. 마이너 업데이트가 가능한 한 오랫동안 이전 버전과 호환되도록 노력하고 있지만, 새로운 마이너 버전이 출시된 후 180일이 지나면 지난 마이너 버전에 대한 지원 종료를 발표할 수 있습니다.

패치 버전: 패치 버전 업데이트는 선택적 버그 수정 및 개선을 위한 것입니다. 최고의 보안 및 사용자 경험을 위해 버전을 최신 상태로 유지하는 것을 권장하지만 새로운 메이저 또는 마이너 버전이 출시될 때까지 패치 업데이트가 이전 버전과 완벽하게 호환되도록 최선을 다하고 있습니다.

버전 관리 기간(메이저의 경우 120일, 마이너의 경우 180일)은 앱의 SDK 업데이트, 앱 스토어 또는 웹사이트에 앱 업로드, 사용자가 최신 버전의 앱을 다운로드하는 데 적용됩니다.

버전 출시 및 호환성 매트릭스

FaceLivenessDetector 구성 요소 또는 사용자 챌린지의 메이저 버전 출시가 겹치는 경우가 종종 있습니다. 버전 종속성을 추적하는 데 도움이 되도록 다음 표에 링크된 리소스를 참조하세요.

SDK 버전 및 changelog:

FaceLivenessDetector for web SDK		FaceLivenessDetector for iOS SDK	FaceLivenessDetector for Android SDK
현재 버전	Changelog	현재 버전/Changelog	현재 버전/Changelog

사용자 챌린지:

Challenge Name	Version	Release date	Retire date
FaceMovementAndLightChallenge	v1.0.0	4/10/2023	N/A
FaceMovementChallenge	v1.0.0	4/30/2025	N/A

새 릴리스에 대한 알림

AWS 는 다음 채널을 통해 새 릴리스를 전달합니다.

- Face Liveness 계정 ID와 연결된 계정 이메일로 서비스 상태 업데이트 이메일 알림 전송.
- 각 GitHub 리포지토리에서 AWS SDKs 및 관련 알림에 대한 업데이트를 게시했습니다.
- 각 GitHub 리포지토리에서 AWS Amplify SDKs 및 관련 알림에 대한 업데이트를 게시했습니다.

이러한 채널을 구독하여 최신 정보를 받아보시기를 권고합니다.

Face Liveness FAQ

다음 FAQ 항목을 사용하여 Rekognition Face Liveness에 관해 자주 묻는 질문에 대한 답변을 찾아보세요.

- 얼굴 생체 확인 검사의 결과물은 무엇입니까?

  Rekognition Face Liveness는 모든 생체 확인 검사에 대해 다음과 같은 출력물을 제공합니다.

  - 신뢰도 점수: 0에서 100 사이의 숫자 점수가 반환됩니다. 이 점수는 셀카 비디오가 스푸핑을 사용한 악의적인 행위자가 아닌 실제 사람이 찍었을 가능성을 나타냅니다.
  - 고품질 이미지: 셀카 비디오에서 고품질 이미지 하나를 추출합니다. 이 프레임은 얼굴 비교, 나이 추정 또는 얼굴 검색과 같은 다양한 용도로 활용할 수 있습니다.
  - 감사 이미지: 셀카 비디오에서 최대 4개의 이미지가 반환되며, 감사 추적 목적으로 사용할 수 있습니다.
- Rekognition Face Liveness는 iBeta Presentation Attack Detection(PAD) 테스트와 호환되나요?

  iBeta Quality Assurance의 Presentation Attack Detection(PAD) 테스트는 ISO/IEC 30107-3에 따라 수행됩니다. iBeta는 NIST/NVLAP의 인증을 받아 이 PAD 표준을 테스트하고 결과를 제공합니다. Rekognition Face Liveness는 PAD 점수 만점으로 레벨 1 및 레벨 2 iBeta Presentation Attack Detection(PAD) 적합성 테스트를 통과했습니다. 보고서는 여기의 iBeta 웹페이지에서 확인할 수 있습니다.
- 고품질 프레임 및 추가 프레임을 어떻게 구할 수 있나요?

  CreateFaceLivenessSession API 요청의 구성에 따라 고품질 프레임 및 추가 프레임을 원시 바이트로 반환하거나 지정한 Amazon S3 버킷에 업로드할 수 있습니다.
- 타원형과 컬러 조명의 위치를 변경할 수 있나요?

  아니요. 타원형 위치와 컬러 조명은 높은 정확도를 위한 기능이므로 사용자 지정할 수 없습니다.
- 애플리케이션에 따라 사용자 인터페이스를 사용자 지정할 수 있나요?

  예. 테마, 색상, 언어, 텍스트 내용, 글꼴 등 대부분의 화면 구성 요소를 애플리케이션에 맞게 사용자 지정할 수 있습니다. 이러한 구성 요소를 사용자 지정하는 방법에 대한 자세한 내용은 React, Swift 및 Android UI 구성 요소 설명서에서 찾을 수 있습니다.
- 카운트다운 시간과 얼굴을 타원형에 맞추는 시간을 사용자 지정할 수 있나요?

  아니요. 카운트다운 시간과 얼굴 맞춤 시간은 높은 정확도 실현과 지연 시간 간 최적의 균형을 제공하기 위해 수천 명의 사용자를 대상으로 한 대규모 내부 연구를 기반으로 사전에 결정된 것입니다. 이러한 이유로 해당 시간 설정은 사용자 지정할 수 없습니다.

- 다양한 컬러 조명이 접근성 가이드라인을 충족하나요?

  예, 당사 제품의 다양한 컬러 조명은 WCAG 2.1에 서술된 접근성 가이드라인을 준수합니다. 수천 번 이상의 사용자 확인을 통해 검증된 바와 같이, 사용자 경험은 초당 약 두 가지 색상을 표시하며, 이는 초당 3가지로 색상을 제한하라는 권장 사항을 준수합니다. 이는 대다수의 사람들에게 간질 발작을 일으킬 가능성을 줄여 줍니다.

- SDK가 최적의 결과를 위해 화면 밝기를 조정하나요?

  Face Liveness mobile SDK(Android 및 iOS용)는 검사가 시작되면 밝기를 자동으로 조정합니다. 하지만 웹 SDK의 경우 자동 밝기 조정을 방지하는 웹 페이지 제한이 있습니다. 이러한 경우 웹 애플리케이션이 최종 사용자에게 최적의 결과를 위해 화면 밝기를 수동으로 높이도록 지시할 것으로 예상합니다.

- 꼭 타원형이어야 하나요? 비슷한 다른 모양을 사용할 수 있을까요?

  아니요, 타원형의 크기, 모양, 위치는 사용자 정의할 수 없습니다. 이 타원형 디자인은 얼굴의 움직임을 정확하게 포착하고 분석하는 데 효과적이기 때문에 신중하게 선택되었습니다. 따라서 타원형 모양은 수정할 수 없습니다.

- 종단간 지연 시간은 어떻게 되나요?

  사용자가 생체 확인 검사를 완료하는 데 필요한 작업을 시작한 시점부터 사용자가 결과(통과 또는 실패)를 얻을 때까지의 전체 지연 시간을 측정합니다. 최상의 경우 지연 시간은 5초입니다. 평균적으로는 약 7초가 될 것으로 예상됩니다. 최악의 경우 지연 시간은 11초입니다. 사용자가 필요한 작업(즉 얼굴을 타원형 안으로 이동하는)을 완료하는 시간, 네트워크 연결, 애플리케이션 지연 시간 등에 따라 종단 간 지연 시간이 달라질 수 있습니다.

- Amplify SDK 없이 Face Liveness 기능을 사용할 수 있나요?

  아니요, Rekognition Face Liveness 기능을 사용하려면 Amplify SDK가 필요합니다.

- Face Liveness와 연결된 오류 상태는 어디에서 찾을 수 있나요?

  여기에서 다양한 Face Liveness 오류 상태를 확인할 수 있습니다.

- 제가 있는 리전에서 Face Liveness를 사용할 수 없습니다. 이 기능을 사용하려면 어떻게 해야 하나요?

트래픽 부하 및 근접성에 따라 Face Liveness를 사용할 수 있는 모든 리전에서 Face Liveness를 직접 호출하도록 할 수 있습니다. 얼굴 생기는 현재 다음 AWS 리전에서 사용할 수 있습니다.

- 미국 동부(버지니아 북부)
- US West (Oregon)
- 유럽(아일랜드)
- 아시아 태평양(도쿄, 뭄바이)

AWS 계정이 다른 리전에 있더라도 지연 시간 차이는 크지 않을 것으로 예상됩니다. Amazon S3 위치를 통해 또는 원시 바이트로 고품질 셀카 프레임 및 감사 이미지를 얻을 수 있지만 Amazon S3 버킷은 Face Liveness AWS 리전과 일치해야 합니다. 리전이 다를 경우 이미지를 원시 바이트로 수신해야 합니다.

- Amazon Rekognition 생체 확인 감지는 서비스 개선을 위해 고객 콘텐츠를 사용하나요?

  AWS Organizations의 옵트아웃 정책을 사용하여 Rekognition 및 기타 Amazon 기계 학습 또는 인공지능 기술의 품질을 개선하거나 개발하는 데 여러분의 이미지 및 비디오 입력을 사용하지 않도록 선택할 수 있습니다. 옵트아웃 방법에 대한 자세한 내용은 AI Services 옵트아웃 정책 관리를 참조하세요.

대량 분석

Amazon Rekognition 대량 분석을 사용하면 StartMediaAnalysisJob 작업으로 매니페스트 파일을 사용하여 대규모 이미지 컬렉션을 비동기적으로 처리할 수 있습니다. 각 개별 이미지의 출력은 분석에 사용한 작업에서 반환된 출력과 일치합니다.

현재 Rekognition은 DetectModerationLabels 작업을 통한 분석을 지원합니다.

작업에서 성공적으로 처리된 이미지 수에 따라 요금이 부과됩니다. 완료된 작업의 결과는 지정한 Amazon S3 버킷으로 출력됩니다.

대량 분석은 Amazon A2I 통합을 지원하지 않는다는 점에 유의하세요.

API는 애니메이션 또는 일러스트 콘텐츠 유형을 감지할 수 있으며, 감지된 콘텐츠 유형에 대한 정보는 응답의 일부로 반환됩니다.

주제

- 대량 이미지 처리
- 대량 분석 출력 매니페스트
- 콘텐츠 유형
- 예측 확인 및 어댑터 훈련

대량 이미지 처리

매니페스트 파일을 제출하고 StartMediaAnalysisJob 작업을 직접적으로 호출하여 새 대량 분석 작업을 시작할 수 있습니다. 입력 매니페스트 파일은 Amazon S3 버킷에 있는 이미지에 대한 참조를 포함하며 형식은 다음과 같습니다.

```
{"source-ref": "s3://foo/bar/1.jpg"}
```

대량 분석 작업(CLI)을 만들려면

1. 아직 설정하지 않았다면 다음과 같이 하세요.

   a. AmazonRekognitionFullAccess 권한과 AmazonS3ReadOnlyAccess 권한을 가진 사용자를 생성하거나 업데이트합니다. 자세한 내용은 1단계: AWS 계정 설정 및 사용자 생성 단원을 참조하십시오.

b. AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. S3 버킷에 이미지를 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

3. 다음 명령을 사용하여 대량 분석 작업을 생성하고 검색하세요.

CLI

DetectModerationLabels 작업을 사용한 분석을 위해 다음 명령을 사용하여 StartMediaAnalysisJob 작업을 직접적으로 호출하세요.

```
# Requests
# Starting DetectModerationLabels job with default settings
aws rekognition start-media-analysis-job \
--operations-config "DetectModerationLabels={MinConfidence='1'}" \
--input "S3Object={Bucket=amzn-s3-demo-source-bucket,Name=my-input.jsonl}" \
--output-config "S3Bucket=amzn-s3-demo-destination-bucket;,S3KeyPrefix=my-results"
```

GetMediaAnalysisJob 작업을 사용하여 결과 및 요약 파일이 저장되는 버킷의 Amazon S3 경로와 같은 특정 작업에 대한 정보를 얻을 수 있습니다. StartMediaAnalysisJob 또는 ListMediaAnalysisJob이 반환한 작업 ID를 제공하면 됩니다. 개별 작업에 대한 세부 정보는 1년 동안만 유지됩니다.

```
# Request
aws rekognition get-media-analysis-job \
--job-id customer-job-id
```

작업 페이지를 반환하는 ListMediaAnalysisJobs 작업을 사용하여 모든 대량 분석의 목록을 나열할 수 있습니다. `max-results` 인수를 사용하면 페이지당 반환할 최대 작업 수를 `max-results`의 값으로 제한하여 지정할 수 있습니다. 페이지당 최대 100개의 결과가 반환됩니다. 개별 작업에 대한 세부 정보는 1년 동안만 유지됩니다.

```
# Request
# Specify number of jobs to return per page, limited to max-results.
aws rekognition list-media-analysis-jobs --max-results 1
```

대량 분석 출력 매니페스트

대량 분석 작업은 작업 결과를 포함하는 출력 매니페스트 파일과 더불어 입력 매니페스트 항목을 처리할 때 발생한 오류에 대한 통계 및 세부 정보가 포함된 매니페스트 요약을 생성합니다.

입력 매니페스트에 중복된 항목이 포함된 경우 작업은 고유한 입력을 필터링하지 않고 대신 제공된 모든 항목을 처리합니다.

출력 매니페스트 파일의 형식은 다음과 같습니다.

```
// 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"}]}}
```

출력 매니페스트 요약의 형식은 다음과 같습니다.

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

콘텐츠 유형

StartMediaAnalysisJob 작업으로 분석된 미디어 콘텐츠 유형에 대한 정보는 GetMediaAnalysisJob 작업으로 반환됩니다. ContentType은 두 가지 범주 중 하나일 수 있습니다.

- 비디오 게임 및 애니메이션(예: 카툰, 만화, 망가, 애니메이션)을 포함한 애니메이션 콘텐츠.
- 그림, 페인팅, 스케치를 포함한 일러스트 콘텐츠.

예측 확인 및 어댑터 훈련

Rekognition 콘솔을 통해 대량 분석을 활용하여 이미지 배치에 대한 예측을 도출하고 이러한 예측을 검증한 후 검증된 예측을 사용하여 어댑터를 생성할 수도 있습니다. 어댑터를 사용하면 지원되는 모든 Rekognition 작업의 정확도를 높일 수 있습니다.

현재 Rekognition Custom Moderation 기능과 함께 사용할 어댑터를 만들 수 있습니다. 어댑터를 만들어 DetectModerationLabels 작업에 제공하면 특정 사용 사례와 관련된 콘텐츠 조절 작업의 정확도를 높일 수 있습니다.

Custom Moderation에 관한 자세한 내용은 사용자 지정 조절을 통한 정확도 향상을 참조하세요.
대량 분석을 통해 예측한 결과를 검증하는 방법에 대한 설명은 대량 분석 및 검증을 참조하세요.
Rekognition 콘솔을 사용하여 예측을 확인하고 어댑터를 만드는 방법을 설명하는 튜토리얼은 사용자 지정 조절 어댑터 자습서을 참조하세요.

자습서

이러한 교차 서비스 자습서에서는 다른 AWS 서비스와 함께 Rekognition의 API 작업을 사용하여 샘플 애플리케이션을 생성하고 다양한 작업을 수행하는 방법을 보여줍니다. 이 자습서의 대부분은 Amazon S3를 사용하여 이미지 또는 비디오를 저장합니다. 일반적으로 사용되는 다른 서비스에는 AWS Lambda가 있습니다.

주제

- Amazon RDS 및 DynamoDB로 Amazon Rekognition 데이터 저장
- Amazon Rekognition과 Lambda를 사용하여 Amazon S3 버킷의 자산에 태그 지정
- AWS 비디오 분석기 애플리케이션 생성
- Amazon Rekognition Lambda 함수 생성
- 신원 확인을 위한 Amazon Rekognition 사용
- Amazon Rekognition(REK)을 사용하여 마케팅 애플리케이션의 레이블 감지
- Lambda와 Python을 사용하여 이미지에서 레이블 감지

Amazon RDS 및 DynamoDB로 Amazon Rekognition 데이터 저장

Amazon Rekognition의 API를 사용할 때는 API 작업에서 생성된 레이블이 저장되지 않는다는 점을 기억해야 합니다. 이러한 레이블을 각 이미지의 식별자와 함께 데이터베이스에 배치하여 저장할 수 있습니다.

이 자습서에서는 레이블을 감지하고 감지된 레이블을 데이터베이스에 저장하는 방법을 보여줍니다. 이 자습서에서 개발한 샘플 애플리케이션은 Amazon S3 버킷에서 이미지를 읽고, 이러한 이미지에 대해 DetectLabels 작업을 직접 호출하고, 결과 레이블을 데이터베이스에 저장합니다. 해당 애플리케이션은 사용하려는 데이터베이스 유형에 따라 Amazon RDS 데이터베이스 인스턴스 또는 DynamoDB 데이터베이스에 데이터를 저장합니다.

이 자습서에서는 AWS SDK for Python을 사용합니다. 더 많은 Python 자습서를 보려면 AWS 설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

주제

- 사전 조건
- Amazon S3 버킷에 있는 이미지에 대한 레이블 가져오기
- Amazon DynamoDB 테이블 생성

- DynamoDB로 데이터 업로드
- Amazon RDS에서 MySQL 데이터베이스 생성
- Amazon RDS MySQL 테이블에 데이터 업로드

사전 조건

이 자습서를 시작하기 전에 Python을 설치하고 Python AWS SDK를 설정하는 데 필요한 단계를 완료해야 합니다. 이 외에도 다음 사항을 반드시 갖추어야 합니다.

AWS 계정 및 IAM 역할 생성

Python SDK(Boto3) 설치

AWS 액세스 자격 증명을 올바르게 구성했습니다.

이미지로 채워진 Amazon S3 버킷 생성

RDS를 사용하여 데이터를 저장하는 경우 RDS 데이터베이스 인스턴스 생성

Amazon S3 버킷에 있는 이미지에 대한 레이블 가져오기

Amazon S3 버킷의 이미지 이름을 가져와서 해당 이미지를 검색하는 함수를 작성하는 것으로 시작하세요. 이 이미지는 함수에 있는 DetectLabels에 대한 직접 호출에 올바른 이미지가 전달되고 있는지 확인하기 위해 표시됩니다.

1. 사용하려는 Amazon S3 버킷을 찾아 이름을 적어 둡니다. 이 Amazon S3 버킷을 직접 호출하여 그 안에 있는 이미지를 읽어올 것입니다. DetectLabels 작업에 전달할 이미지가 버킷에 포함되어 있는지 확인하세요.
2. Amazon S3 버킷에 연결하기 위한 코드를 작성합니다. Boto3를 사용하여 Amazon S3 리소스에 연결하고 Amazon S3 버킷에서 이미지를 검색할 수 있습니다. Amazon S3 리소스에 연결되면 Bucket 메서드에 Amazon S3 버킷 이름을 제공하여 버킷에 액세스할 수 있습니다. Amazon S3 버킷에 연결한 후 Object 메서드를 사용하여 버킷에서 이미지를 가져옵니다. Matplotlib를 사용하면 이 연결을 사용하여 이미지가 처리되는 모습을 시각화할 수 있습니다. Boto3는 Rekognition 클라이언트에 연결하는 데에도 사용됩니다.

   다음 코드에서 region_name 파라미터에 리전을 입력하세요. Amazon S3 버킷 이름과 이미지 이름을 DetectLabels에 전달하면 해당 이미지의 레이블이 반환됩니다. 응답에서 레이블만 선택하면 이미지 이름과 레이블이 모두 반환됩니다.

```
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. 이 코드를 get_images.py 파일에 저장합니다.

Amazon DynamoDB 테이블 생성

다음 코드는 Boto3를 사용하여 DynamoDB에 연결하고 DynamoDB `CreateTable` 메서드를 사용하여 Images라는 테이블을 생성합니다. 이 테이블은 Image라는 파티션 키와 Labels라는 정렬 키로 구성된 복합 기본 키를 갖습니다. Image 키에는 이미지 이름이 포함되고 Labels 키에는 해당 Image에 할당된 레이블이 저장됩니다.

```
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)
```

이 코드를 편집기에 저장하고 한 번 실행하여 DynamoDB 테이블을 생성합니다.

DynamoDB로 데이터 업로드

이제 DynamoDB 데이터베이스가 생성되고 이미지의 레이블을 가져오는 함수가 생겼으므로 DynamoDB에 레이블을 저장할 수 있습니다. 다음 코드는 S3 버킷의 모든 이미지를 검색하고 이미지에 대한 레이블을 가져온 다음 DynamoDB에 데이터를 저장합니다.

1. DynamoDB에 데이터를 업로드하기 위한 코드를 작성해야 합니다. `get_image_names`라는 함수는 Amazon S3 버킷에 연결하는 데 사용되며 버킷에 있는 모든 이미지의 이름을 목록으로 반환합니다. 이 목록을 생성한 `get_images.py` 파일에서 가져온 `read_image_from_S3` 함수에 전달합니다.

   ```
   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. 앞서 만든 `read_image_from_S3` 함수는 처리 중인 이미지의 이름과 해당 이미지와 연결된 레이블 사전을 반환합니다. `find_values`라는 함수는 응답에서 레이블만을 가져오는 데 사용됩니다. 그러면 이미지 이름과 레이블을 DynamoDB 테이블에 업로드할 준비가 완료됩니다.

   ```
   def find_values(id, json_repr):
       results = []

       def _decode_dict(a_dict):
           try:
               results.append(a_dict[id])
   ```

```
        except KeyError:
            pass
        return a_dict

    json.loads(json_repr, object_hook=_decode_dict) # Return value ignored.
    return results
```

3. load_data라는 세 번째 함수를 사용하여 생성한 DynamoDB 테이블에 이미지와 레이블을 실제로 로드합니다.

```
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. 여기서 이전에 정의한 세 가지 함수가 직접 호출되고 작업이 수행됩니다. 위에서 정의한 세 가지 함수를 아래 코드와 함께 Python 파일에 추가합니다. 코드를 실행합니다.

```
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!")
```

DetectLabels를 사용하여 이미지에 대한 레이블을 생성하고 해당 레이블을 DynamoDB 인스턴스에 저장했습니다. 이 자습서를 진행하면서 생성한 리소스를 모두 제거했는지 반드시 확인하세요. 그러면 사용하지 않는 리소스에 대한 비용이 청구되는 것을 방지할 수 있습니다.

Amazon RDS에서 MySQL 데이터베이스 생성

계속 진행하기 전에 Amazon RDS의 설정 절차를 완료하고 Amazon RDS를 사용하여 MySQL DB 인스턴스를 생성했는지 확인하세요.

다음 코드는 PyMySQL 라이브러리와 Amazon RDS DB 인스턴스를 사용합니다. 이는 이미지 이름 및 해당 이미지와 연결된 레이블을 보관할 테이블을 생성합니다. Amazon RDS는 테이블을 생성하고 테이블에 데이터를 삽입하라는 명령을 받습니다. Amazon RDS를 사용하려면 호스트 이름, 사용자 이름 및 암호를 사용하여 Amazon RDS 호스트에 연결해야 합니다. PyMySQL의 `connect` 함수에 이러한 인수를 제공하고 커서 인스턴스를 생성하여 Amazon RDS에 연결합니다.

1. 다음 코드에서는 호스트의 값을 Amazon RDS 호스트 엔드포인트로 바꾸고, 사용자의 값을 Amazon RDS 인스턴스와 연결된 마스터 사용자 이름으로 바꿉니다. 또한 암호를 기본 사용자의 마스터 암호로 바꿔야 합니다.

```
import pymysql

host = "host-endpoint"
user = "username"
password = "master-password"
```

2. 이미지와 레이블 데이터를 삽입할 데이터베이스와 테이블을 만드세요. 생성 쿼리를 실행하고 커밋하면 이 작업을 수행할 수 있습니다. 다음 코드는 데이터베이스를 생성합니다. 이 코드는 한 번만 실행하세요.

```
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. 데이터베이스를 만든 후에는 이미지 이름과 레이블을 삽입할 테이블을 만들어야 합니다. 테이블을 생성하려면 먼저 use SQL 명령을 데이터베이스 이름과 함께 `execute` 함수에 전달해야 합니다. 연결이 완료되면 테이블을 만들기 위한 쿼리가 실행됩니다. 다음 코드는 데이터베이스에 연결한 다음 `image_id`라는 이름의 프라이머리 키와 레이블을 저장하는 텍스트 속성을 모두 포함하는 테이블을 만듭니다. 앞서 정의한 임포트와 변수를 사용하고 이 코드를 실행하여 데이터베이스에 테이블을 생성합니다.

```
# connect to existing DB
cursor.execute("use rekogDB1")
cursor.execute("CREATE TABLE IF NOT EXISTS test_table(image_id VARCHAR (255)
 PRIMARY KEY, image_labels TEXT)")
conn.commit()
print("Table creation - Successful creation!")
```

Amazon RDS MySQL 테이블에 데이터 업로드

Amazon RDS 데이터베이스를 생성하고 그 안에 테이블을 생성하고 나면 이미지의 레이블을 가져와서 Amazon RDS 데이터베이스에 해당 레이블을 저장할 수 있습니다.

1. Amazon S3 버킷에 연결하고 버킷에 있는 모든 이미지의 이름을 검색합니다. 이러한 이미지 이름은 이전에 생성한 `read_image_from_s3` 함수에 전달되어 모든 이미지의 레이블을 가져옵니다. 다음 코드는 Amazon S3 버킷에 연결하고 버킷 안의 모든 이미지 목록을 반환합니다.

```
import pymysql
from get_images import read_image_from_s3
import json
import boto3

host = "host-endpoint"
user = "username"
password = "master-password"

conn = pymysql.connect(host=host, user=user, passwd=password)
print(conn)
cursor = conn.cursor()
print("Connection successful")

def get_image_names(name_of_bucket):

    s3_resource = boto3.resource('s3')
```

```
    my_bucket = s3_resource.Bucket(name_of_bucket)
    file_list = []
    for file in my_bucket.objects.all():
        file_list.append(file.key)
    return file_list
```

2. DetectLabels API의 응답에는 레이블만 포함된 것이 아니므로 레이블 값만 추출하는 함수를 작성하세요. 다음 함수는 레이블만 포함된 목록을 반환합니다.

```
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. 테이블에 이미지 이름과 레이블을 삽입하는 함수가 필요합니다. 다음 함수는 삽입 쿼리를 실행하고 주어진 이미지 이름과 레이블 쌍을 삽입합니다.

```
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. 마지막으로 위에서 정의한 함수를 실행해야 합니다. 다음 코드에서는 버킷에 있는 모든 이미지의 이름을 수집하여 DetectLabels를 직접 호출하는 함수에 제공합니다. 그런 다음 레이블과 해당 레이블이 적용되는 이미지 이름이 Amazon RDS 데이터베이스에 업로드됩니다. 위에서 정의한 세 가지 함수를 아래 코드와 함께 복사하여 Python 파일에 추가합니다. Python 파일을 실행합니다.

```
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!")
```

DetectLabels를 사용하여 이미지에 대한 레이블을 성공적으로 생성하고 Amazon RDS를 사용하여 해당 레이블을 MySQL 데이터베이스에 저장했습니다. 이 자습서를 진행하면서 생성한 리소스를 모두 제거했는지 반드시 확인하세요. 그러면 사용하지 않는 리소스에 대한 비용이 청구되는 것을 방지할 수 있습니다.

AWS 다중 서비스 예제에 대한 자세한 내용은 AWS 설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

Amazon Rekognition과 Lambda를 사용하여 Amazon S3 버킷의 자산에 태그 지정

이 자습서에서는 Amazon S3 버킷에 있는 디지털 자산에 자동으로 태그를 지정하는 AWS Lambda함수를 생성합니다. Lambda 함수는 지정된 Amazon S3 버킷에서 모든 객체를 읽습니다. 버킷의 각 객체에 대해 Amazon Rekognition 서비스로 이미지를 전달하여 일련의 레이블을 생성합니다. 각 레이블은 이미지에 적용되는 태그를 생성하는 데 사용됩니다. Lambda 함수를 실행하면 주어진 Amazon S3 버킷의 모든 이미지를 기반으로 태그를 자동으로 생성하여 이미지에 적용합니다.

예를 들어, Lambda 함수를 실행하고 Amazon S3 버킷에 이 이미지가 있다고 가정해 보겠습니다.

그러면 애플리케이션이 자동으로 태그를 생성하여 이미지에 적용합니다.

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**
>
> 이 자습서에서 사용하는 서비스는 AWS프리 티어의 일부입니다. 자습서를 모두 마치면 요금이 부과되지 않도록 자습서를 따라하는 중에 만든 모든 리소스를 종료하는 것을 권장합니다.

이 자습서에서는 AWSSDK for Java 버전 2를 사용합니다. 추가 Java V2 자습서를 보려면 AWS 설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

주제

- 사전 조건
- IAM Lambda 역할 구성
- 프로젝트 생성
- 코드 쓰기

- 프로젝트 패키지화
- Lambda 함수 배포
- Lambda 메서드 테스트

사전 조건

시작하기 전에 Java용 AWSSDK 설정의 단계를 완료해야 합니다. 그 후 다음 항목들이 있는지 확인하세요.

- Java 1.8 JDK
- Maven 3.6 이상
- 5~7개의 자연 이미지가 들어 있는 Amazon S3 버킷 Lambda 함수가 이 이미지를 읽습니다.

IAM Lambda 역할 구성

이 자습서에서는 Amazon Rekognition 및 Amazon S3 서비스를 사용합니다. Lambda 함수에서 이러한 서비스를 간접적으로 호출할 수 있는 정책을 포함하도록 lambda-support 역할을 구성합니다.

역할을 구성하려면

1. 에 로그인AWS Management Console하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. AWS 서비스와 Lambda를 차례대로 선택합니다.
4. 권한 탭을 선택합니다.
5. AWSLambdaBasicExecutionRole을 검색합니다.
6. 다음 태그를 선택합니다.
7. 검토를 선택합니다.
8. 역할 이름을 lambda-support로 지정합니다.
9. 역할 생성을 선택합니다.
10. lambda-support를 선택하여 개요 페이지를 봅니다.
11. 정책 연결을 선택합니다.
12. 정책 목록에서 AmazonRekognitionFullAccess를 선택합니다.

13. 정책 연결을 선택합니다.
14. AmazonS3FullAccess를 검색한 다음 정책 연결을 선택합니다.

프로젝트 생성

새 Java 프로젝트를 생성한 다음 필요한 설정과 종속성을 사용하여 Maven pom.xml를 구성합니다. pom.xml 파일이 다음과 같은지 확인합니다.

```
<?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>
```

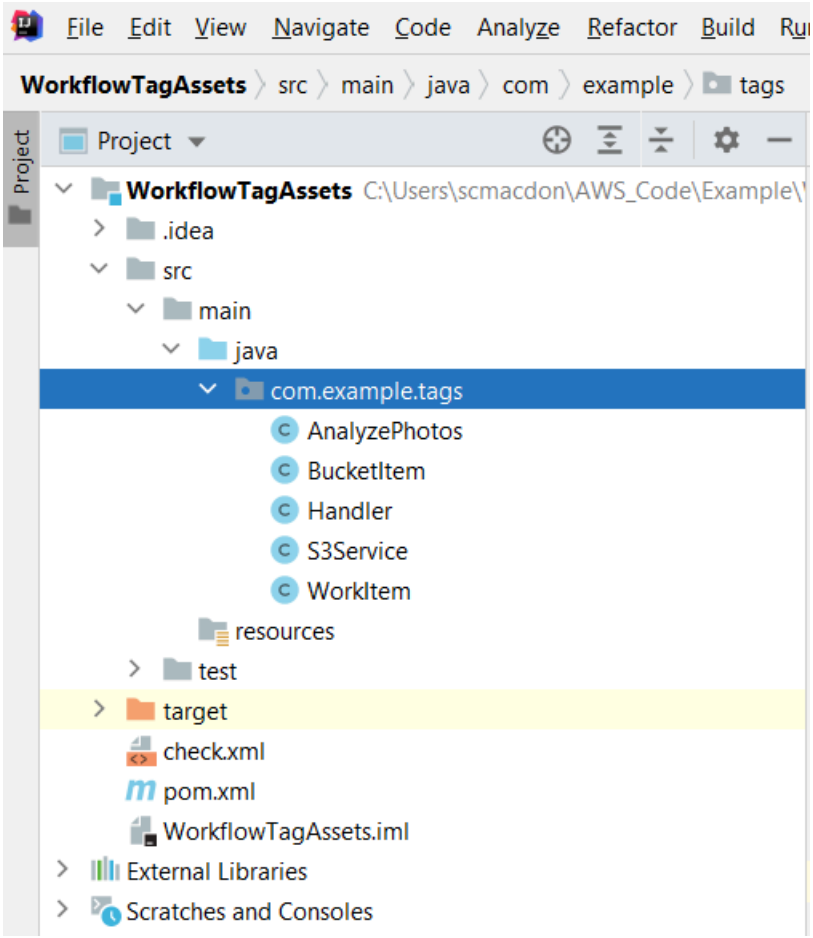
```
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.10.0</version>
    </dependency>
    <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>
```

코드 쓰기

AWS Lambda런타임 Java API를 사용하여 Lambda 함수를 정의하는 Java 클래스를 생성합니다. 이 예제에는 Handler라는 Lambda 함수의 Java 클래스 1개와 이 사용 사례에 필요한 추가 클래스

가 있습니다. 다음 그림은 이 프로젝트의 Java 클래스를 보여 줍니다. 참고로 모든 Java 클래스는 com.example.tags라는 이름의 패키지에 들어 있습니다.



코드에 사용할 다음과 같은 Java 클래스를 생성합니다.

- 핸들러는 Lambda Java 런타임 API를 사용하고이 AWS자습서에 설명된 사용 사례를 수행합니다. 실행되는 애플리케이션 로직은 handleRequest 메서드에 있습니다.
- S3Service는 Amazon S3 API를 사용하여 S3 작업을 수행합니다.
- AnalyzePhotos는 Amazon Rekognition API를 사용하여 이미지를 분석합니다.
- BucketItem은 Amazon S3 버킷 정보를 저장하는 모델을 정의합니다.
- WorkItem은 Amazon Rekognition 데이터를 저장하는 모델을 정의합니다.

Handler 클래스

이 Java 코드는 Handler 클래스를 나타냅니다. 클래스는 Lambda 함수에 전달된 플래그를 읽습니다. S3Service.ListBucketObjects 메서드는 List 객체를 반환하며 여기서 각 요소는 객체 키를 나타내는 문

자열 값입니다. 플래그 값이 true인 경우 s3Service.tagAssets 메서드를 직접 호출하여 목록을 반복하고 태그를 각각의 객체에 적용함으로써 태그를 적용합니다. 플래그 값이 false이면 태그를 삭제하는 s3Service.deleteTagFromObject 메서드가 간접적으로 호출됩니다. 또한 LambdaLogger 객체를 사용하여 Amazon CloudWatch 로그에 메시지를 기록할 수 있다는 점도 참고하세요.

> **Note**
>
> 반드시 bucketName 변수에 버킷 이름을 할당해야 합니다.

```
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 클래스

다음 클래스는 Amazon S3 API를 사용하여 S3 작업을 수행합니다. 예를 들어 getObjectBytes 메서드는 이미지를 나타내는 바이트 배열을 반환합니다. 마찬가지로, listBucketObjects 메서드는 List 객체를 반환하며 여기서 각 요소는 키 이름을 나타내는 문자열 값입니다.

```
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 클래스

다음 Java 코드는 AnalyzePhotos 클래스를 나타냅니다. 이 클래스는 Amazon Rekognition API를 사용하여 이미지를 분석합니다.

```
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 클래스

다음 Java 코드는 Amazon S3 객체 데이터를 저장하는 BucketItem 클래스를 나타냅니다.

```
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 클래스

다음 Java 코드는 WorkItem 클래스를 나타냅니다.

```
 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;
```

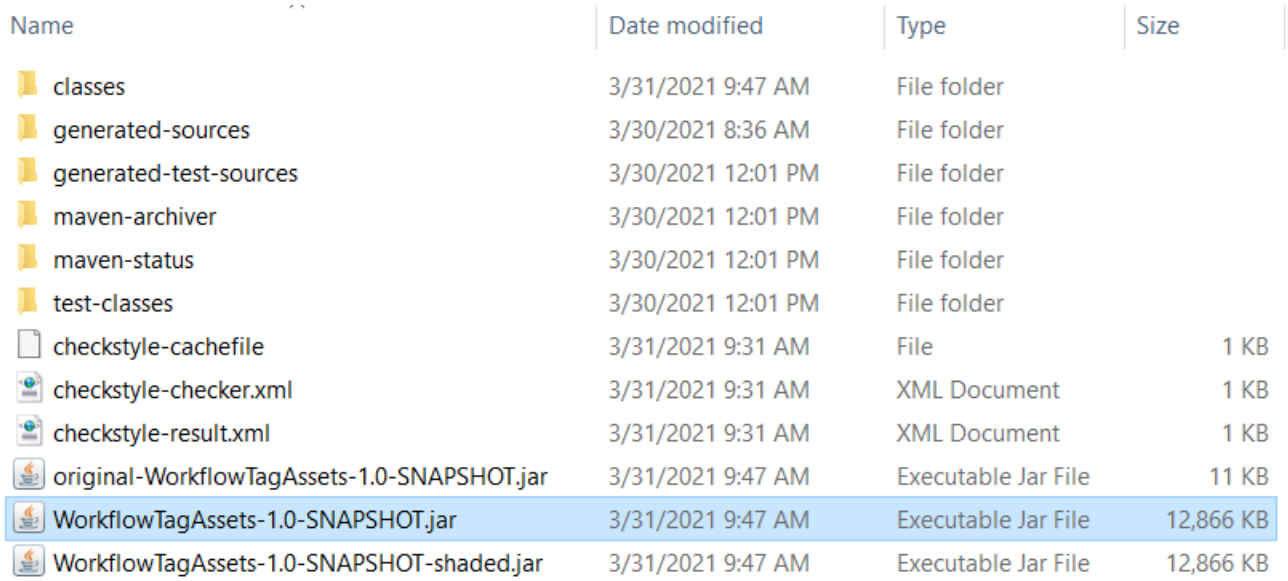
```
}

public String getName() {
    return this.name;
}

public void setConfidence (String confidence) {
    this.confidence = confidence;
}

public String getConfidence() {
    return this.confidence;
}

}
```

프로젝트 패키지화

다음 Maven 명령을 사용하여 프로젝트를.jar(JAR) 파일로 패키징합니다.

```
mvn package
```

JAR 파일은 대상 폴더(프로젝트 폴더의 하위 폴더)에 있습니다.



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**
>
> 프로젝트의 POM 파일에 maven-shade-plugin 사용된 것을 볼 수 있습니다. 이 플러그인은 필수 종속성을 포함하는 JAR을 만드는 역할을 합니다. 이 플러그인 없이 프로젝트를 패키지화하려고 하면 필수 종속성이 JAR 파일에 포함되지 않아 ClassNotFoundException이 발생합니다.

Lambda 함수 배포

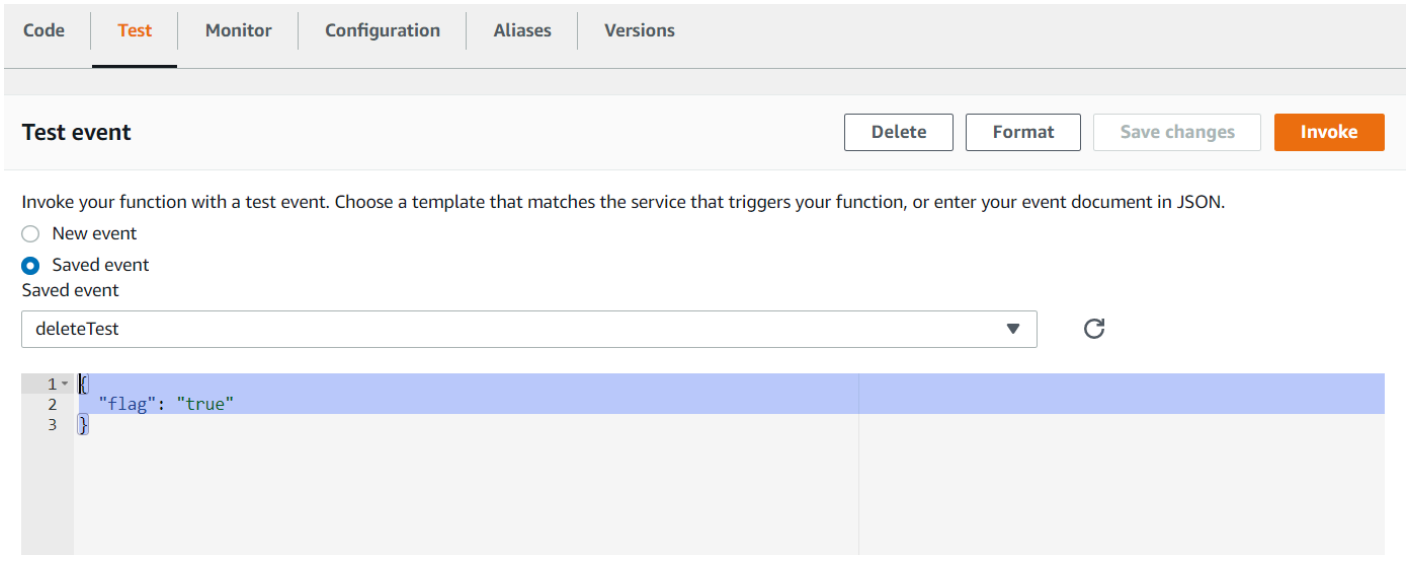
1. Lambda 콘솔을 엽니다.
2. 함수 생성을 선택합니다.
3. 새로 작성을 선택합니다.
4. 기본 정보 섹션에서 이름으로 cron을 입력합니다.
5. 런타임에서 Java 8을 선택합니다.
6. 기존 역할 사용을 선택한 다음, lambda-support(생성한 IAM 역할)를 선택합니다.
7. 함수 생성을 선택합니다.
8. 코드 항목 유형에서 .zip 또는 .jar 파일 업로드를 선택합니다.
9. 업로드를 선택한 다음 생성한 JAR 파일을 찾아보세요.
10. 핸들러의 경우 정규화된 함수의 전체 이름, 예를 들어 com.example.tags.Handler:handleRequest(com.example.tags는 패키지를 지정하고, 핸들러는 :: 및 메서드 이름 앞에 오는 클래스입니다)를 입력합니다.
11. 저장을 선택합니다.

Lambda 메서드 테스트

자습서의 이 시점에서 Lambda 함수를 테스트할 수 있습니다.

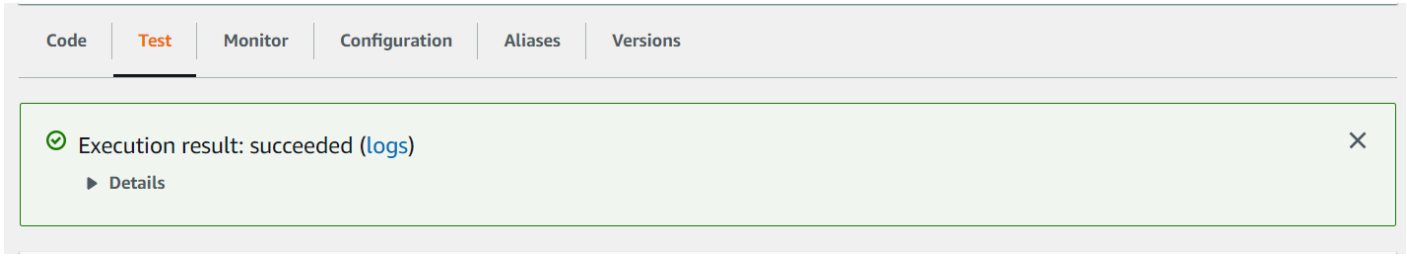
1. Lambda 콘솔에서 테스트 탭을 클릭한 후 다음 JSON을 입력합니다.

```
                {
"flag": "true"
 }
```



> **Note**
>
> true 태그를 전달하면 디지털 자산에 태그를 지정하고 false를 전달하면 태그가 삭제됩니다.

2. 간접 호출 버튼을 선택합니다. Lambda 함수가 간접 호출되면 성공 메시지를 확인할 수 있습니다.



축하합니다. Amazon S3 버킷에 있는 디지털 자산에 태그를 자동으로 적용하는 AWS Lambda함수를 생성했습니다. 이 자습서의 시작 부분에서 설명한 것처럼, 요금이 부과되지 않도록 하려면 이 자습서를 진행하는 동안 생성한 모든 리소스를 종료해야 합니다.

자세한 AWS다중 서비스 예제는 AWS설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

AWS 비디오 분석기 애플리케이션 생성

AWS SDK for Java version 2를 사용하여 비디오를 분석해 레이블을 감지하는 Java 웹 애플리케이션을 만들 수 있습니다. 이 AWS 자습서에서 만든 애플리케이션을 사용하면 Amazon S3 버킷에 비디오(MP4 파일)를 업로드할 수 있습니다. 그런 다음 애플리케이션이 Amazon Rekognition 서비스를 사용하

여 비디오를 분석합니다. 그 결과를 사용하여 데이터 모델을 채운 다음 Amazon Simple Email Service를 사용하여 보고서를 생성하고 특정 사용자에게 이메일로 보냅니다.

다음 그림은 애플리케이션이 비디오 분석을 완료한 후 생성되는 보고서를 보여줍니다. 아래 표의 열에는 Age Range, Beard, Eye glasses, Eyes open과 다양한 속성 예측에 대한 신뢰도 값이 표시됩니다.

	Age Range	Beard	Eye glasses	Eyes open
2				
3	AgeRange(Low=38, High=56)	Beard(Value=false, Confidence=83.07253)	Eyeglasses(Value=true, Confidence=55.965977)	EyeOpen(Value=true, Confidence=94.691696)
4	AgeRange(Low=36, High=52)	Beard(Value=true, Confidence=50.721912)	Eyeglasses(Value=false, Confidence=63.886036)	EyeOpen(Value=true, Confidence=95.906364)
5	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=58.38352)	Eyeglasses(Value=false, Confidence=96.39576)	EyeOpen(Value=true, Confidence=53.580643)
6	AgeRange(Low=49, High=67)	Beard(Value=false, Confidence=81.41662)	Eyeglasses(Value=true, Confidence=65.28722)	EyeOpen(Value=true, Confidence=95.11523)
7	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=61.533833)	Eyeglasses(Value=false, Confidence=97.51163)	EyeOpen(Value=true, Confidence=82.21834)
8	AgeRange(Low=29, High=45)	Beard(Value=false, Confidence=74.22591)	Eyeglasses(Value=true, Confidence=64.906685)	EyeOpen(Value=true, Confidence=98.48175)
9	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=65.9394)	Eyeglasses(Value=false, Confidence=94.14824)	EyeOpen(Value=true, Confidence=94.857346)
10	AgeRange(Low=44, High=62)	Beard(Value=true, Confidence=78.648)	Eyeglasses(Value=true, Confidence=65.83134)	EyeOpen(Value=true, Confidence=98.538666)
11				

이 자습서에서는 다양한 AWS 서비스를 간접 호출하는 Spring Boot 애플리케이션을 생성합니다. Spring Boot API는 모델과 다양한 뷰 및 컨트롤러를 구축하는 데 사용됩니다. 자세한 내용은 Spring Boot를 참조하세요.

이 서비스는 다음 AWS 서비스를 사용합니다.

- Amazon Rekognition
- Amazon S3()
- Amazon SES
- AWS Elastic Beanstalk

이 자습서에 포함된 AWS 서비스는 AWS 프리 티어에 포함됩니다. 요금이 부과되지 않도록 하기 위해 자습서 완료 후 자습서 진행 중에 만든 모든 리소스를 종료하는 것을 권장합니다.

사전 조건

시작하기 전에 AWS SDK for Java 설정에 나온 단계를 완료해야 합니다. 그 후 다음 항목들이 있는지 확인하세요.

- Java 1.8 JDK
- Maven 3.6 이상
- video[somevalue]라는 이름의 Amazon S3 버킷 Amazon S3 Java 코드에 이 버킷 이름을 사용해야 합니다. 자세한 내용은 버킷 생성 단원을 참조하세요.
- IAM 역할. 생성할 VideoDetectFaces 클래스에 필요한 항목입니다. 자세한 내용은 Amazon Rekognition Video 설정을 참조하세요.

- 유효한 Amazon SNS 주제. 생성할 VideoDetectFaces 클래스에 필요한 항목입니다. 자세한 내용은 Amazon Rekognition Video 설정을 참조하세요.

절차

이 자습서를 진행하면서 다음 작업을 수행합니다.

1. 프로젝트 생성
2. 프로젝트에 POM 종속 항목 추가
3. Java 클래스 생성
4. HTML 파일 생성
5. 스크립트 파일 생성
6. 프로젝트를 JAR 파일로 패키징
7. 애플리케이션을 AWS Elastic Beanstalk에 배포

자습서를 계속 진행하려면 AWS 설명서 SDK 예제 GitHub 리포지토리의 자세한 지침을 따르세요.

Amazon Rekognition Lambda 함수 생성

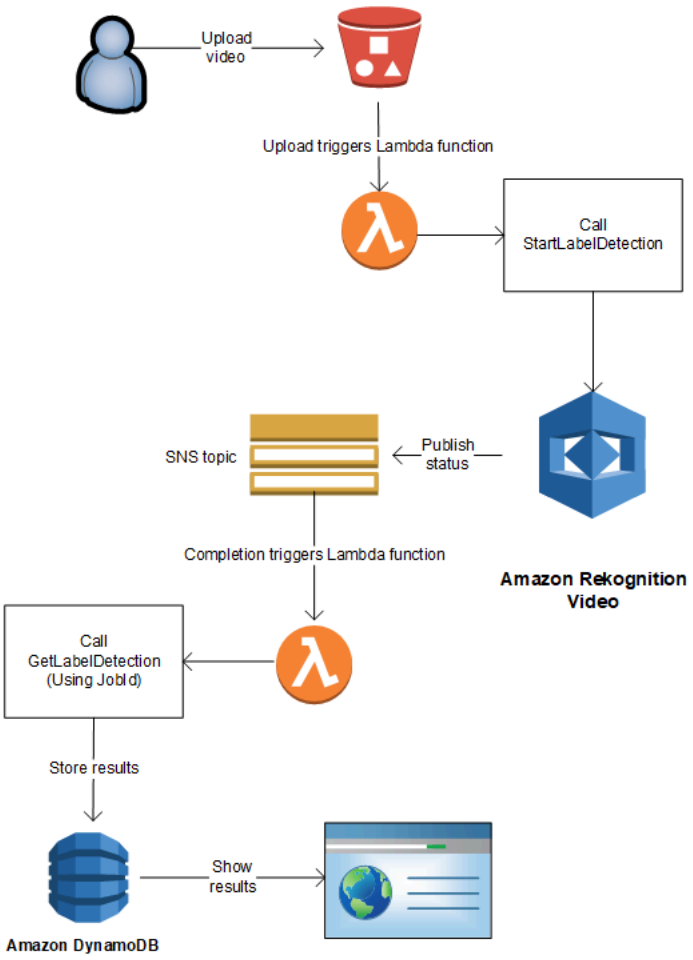
이 자습서에서는 Java Lambda 함수를 사용하여 비디오 분석 작업의 결과를 레이블 감지를 위해 가져오는 방법을 보여줍니다.

> **Note**
>
> 이 자습서에서는 Java 1.x용 AWS SDK를 사용합니다. Rekognition 및 AWS SDK for Java 버전 2를 사용하는 자습서는 AWS 설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

Amazon Rekognition Video 작업과 함께 Lambda 함수를 사용할 수 있습니다. 예를 들어 다음 다이어그램은 Amazon S3 버킷에 업로드되면 Lambda 함수를 사용하여 비디오 분석을 자동으로 시작하는 웹사이트를 보여줍니다. Lambda 함수는 트리거되면 StartLabelDetection을 직접 호출하여 업로드된 비디오에서 레이블 감지를 시작합니다. Lambda를 사용하여 Amazon S3 버킷의 이벤트 알림을 처리하는 방법에 대한 자세한 내용은 Amazon S3 이벤트에서 AWS Lambda 사용을 참조하세요.

두 번째 Lambda 함수는 분석 완료 상태가 등록된 Amazon SNS 주제에 전송되면 트리거됩니다. 두 번째 Lambda 함수는 GetLabelDetection을 직접 호출하여 분석 결과를 가져옵니다. 그러면 결과가 데이

터베이스에 저장되어 웹 사이트에 표시될 준비가 됩니다. 이 두 번째 Lambda 함수가 이 자습서의 초점입니다.



이 자습서에서는 Amazon Rekognition Video가 등록된 Amazon SNS 주제로 비디오 분석의 완료 상태를 보내면 Lambda 함수가 트리거됩니다. 그런 다음 GetLabelDetection을 직접 호출하여 비디오 분석 결과를 수집합니다. 설명을 위해 이 자습서에서는 레이블 감지 결과를 CloudWatch 로그에 기록합니다. 애플리케이션의 Lambda 함수에서 나중에 사용할 수 있도록 분석 결과를 저장해야 합니다. 예를 들어 Amazon DynamoDB를 사용하여 분석 결과를 저장할 수 있습니다. 자세한 내용은 DynamoDB 사용 단원을 참조하십시오.

다음 절차에서는 이 방법을 보여 줍니다.

- Amazon SNS 주제를 만들고 권한을 설정합니다.
- 를 사용하여 Lambda 함수를 생성하고 Amazon SNS 주제를 AWS Management Console 구독합니다.
- AWS Management Console을 사용하여 Lambda 함수를 구성합니다.
- AWS Toolkit for Eclipse 프로젝트에 샘플 코드를 추가하고 Lambda 함수에 업로드합니다.

- AWS CLI를 사용하여 Lambda 함수를 테스트합니다.

> **Note**
>
> 자습서 전체에서 동일한 AWS 리전을 사용합니다.

사전 조건

이 자습서에서는 사용자가 AWS Toolkit for Eclipse를 잘 알고 있는 것으로 가정합니다. 자세한 내용은 AWS Toolkit for Eclipse를 참조하십시오.

SNS 주제 생성

Amazon Rekognition Video 비디오 분석 작업의 완료 상태는 Amazon SNS 주제에 전송됩니다. 이 절차를 통해 Amazon SNS 주제 및 Amazon Rekognition Video에 Amazon SNS 주제에 대한 액세스 권한을 부여하는 IAM 서비스 역할이 생성됩니다. 자세한 내용은 Amazon Rekognition Video 작업 직접 호출 단원을 참조하십시오.

Amazon SNS 토픽을 생성하려면

1. 아직 생성하지 않았다면 Amazon Rekognition Video에 Amazon SNS 주제에 대한 액세스 권한을 부여할 IAM 서비스 역할을 생성합니다. Amazon 리소스 이름(ARN)을 적어 둡니다. 자세한 내용은 여러 Amazon SNS 주제에 대한 액세스 권한 부여 단원을 참조하십시오.
2. Amazon SNS 콘솔을 사용하여 Amazon SNS 주제를 생성합니다. 주제 이름만 지정하면 됩니다. 주제 이름에 AmazonRekognition을 추가합니다. 주제 ARN을 기록합니다.

Lambda 함수 생성

AWS Management Console을 사용하여 Lambda 함수를 생성합니다. 그런 다음 AWS Toolkit for Eclipse 프로젝트를 사용하여 AWS Lambda에 Lambda 함수 패키지를 업로드합니다. AWS Toolkit for Eclipse로 Lambda 함수를 생성할 수도 있습니다. 자세한 내용은 자습서: AWS Lambda 함수 생성, 업로드 및 호출 방법 단원을 참조하십시오.

Lambda 함수를 만들려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/lambda/에서 AWS Lambda 콘솔을 엽니다.

2. 함수 생성을 선택합니다.
3. 새로 작성을 선택합니다.
4. 함수 이름에 함수의 이름을 입력합니다.
5. 런타임에서 Java 8을 선택합니다.
6. Choose or create an execution role(실행 역할 선택 또는 생성)을 선택합니다.
7. Execution role(실행 역할)에서 Create a new role with basic Lambda permissions(기본 Lambda 권한을 가진 새 역할 생성)를 선택합니다.
8. 기본 정보 섹션의 맨 아래에 표시되는 새 역할의 이름을 확인합니다.
9. 함수 생성을 선택합니다.

Lambda 함수 구성

Lambda 함수를 생성한 후 SNS 주제 생성에서 생성하는 Amazon SNS 주제에 의해 트리거되도록 구성합니다. Lambda 함수의 메모리 요구 사항 및 제한 시간도 조정합니다.

Lambda 함수를 구성하려면

1. 함수 코드에 핸들러로 `com.amazonaws.lambda.demo.JobCompletionHandler`를 입력합니다.
2. Basic settings(기본 설정)에서 편집을 선택합니다. Edit basic settings(기본 설정 편집) 대화 상자가 표시됩니다.

   a. 메모리로 1024를 선택합니다.

   b. 제한 시간으로 10초를 선택합니다.

   c. 저장을 선택합니다.
3. Designer에서 +트리거 추가를 선택합니다. 트리거 추가 대화 상자가 표시됩니다.
4. Trigger configuration(트리거 구성)에서 SNS를 선택합니다.

   SNS 주제에서, SNS 주제 생성에서 생성한 Amazon SNS 주제를 선택합니다.
5. 트리거 활성화를 선택합니다.
6. 트리거를 추가하려면 추가를 선택합니다.
7. 저장을 선택하여 Lambda 함수를 저장합니다.

IAM Lambda 역할 구성

Amazon Rekognition Video 작업을 직접 호출하려면 AmazonRekognitionFullAccess AWS 관리형 정책을 IAM Lambda 역할에 추가합니다. StartLabelDetection 등의 시작 작업에도 Amazon Rekognition Video가 Amazon SNS 주제에 액세스하는 데 사용하는 IAM 서비스 역할에 대한 역할 전달 권한이 필요합니다.

역할을 구성하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. Lambda 함수 생성에서 생성한 실행 역할의 이름을 목록에서 선택합니다.
4. 권한(Permissions) 탭을 선택합니다.
5. 정책 연결(Attach policies)을 선택합니다.
6. 정책 목록에서 AmazonRekognitionFullAccess를 선택합니다.
7. 정책 연결을 선택합니다.
8. 실행 역할을 다시 선택합니다.
9. 인라인 정책 추가를 선택합니다.
10. JSON 탭을 선택합니다.
11. 기존 정책을 다음 정책으로 바꿉니다. `servicerole`을 SNS 주제 생성에서 생성한 IAM 서비스 역할로 바꿉니다.
12. 정책 검토를 선택합니다.
13. 이름*에 정책 이름을 입력합니다.
14. 정책 생성을 선택합니다.

AWS Toolkit for Eclipse Lambda 프로젝트 생성

Lambda 함수가 트리거되면 다음 코드가 Amazon SNS 주제에서 완료 상태를 가져오고 GetLabelDetection을 직접 호출하여 분석 결과를 가져옵니다. 감지된 레이블 수와 감지된 레이블 목록이 CloudWatch 로그에 기록됩니다. Lambda 함수는 나중에 사용할 수 있도록 비디오 분석 결과를 저장해야 합니다.

AWS Toolkit for Eclipse Lambda 프로젝트를 생성하려면

1. AWS Toolkit for EclipseAWS Lambda 프로젝트를 생성합니다.
   - 프로젝트 이름:에 선택한 프로젝트 이름을 입력합니다.
   - Class Name(클래스 이름):에 JobCompletionHandler를 입력합니다.
   - 입력 유형:에서 SNS 이벤트를 선택합니다.
   - 다른 필드는 바꾸지 않고 그대로 둡니다.
2. Eclipse 프로젝트 탐색기에서 생성된 Lambda 핸들러 메서드(JobCompletionHandler.java)를 열고 내용을 다음으로 바꿉니다.

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

package com.amazonaws.lambda.demo;

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



public class JobCompletionHandler implements RequestHandler<SNSEvent, String> {

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

      String message = event.getRecords().get(0).getSNS().getMessage();
```

```
      LambdaLogger logger = context.getLogger();

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

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

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

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

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


      }

      return message;
   }

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

      LambdaLogger logger = context.getLogger();

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

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

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

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

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

         labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

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

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

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

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

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

   }


}
```

3. Rekognition 네임스페이스가 해결되지 않았습니다. 이를 수정하려면:

   - `import com.amazonaws.services.rekognition.AmazonRekognition;` 행의 밑줄이 표시된 부분 위에서 마우스를 멈춥니다.
   - Fix project set up...(프로젝트 설정 수정...)을 선택합니다.
   - Amazon Rekognition 아카이브의 최신 버전을 선택합니다.
   - 확인을 선택하여 프로젝트에 아카이브를 추가합니다.

4. 파일을 저장합니다.
5. Eclipse 코드 창에서 마우스 오른쪽 버튼을 클릭하고 [AWS Lambda]와 [Upload function to AWS Lambda]를 차례대로 선택합니다.
6. [Select Target Lambda Function] 페이지에서 사용할 AWS 리전을 선택합니다.
7. Choose an existing lambda function(기존 lambda 함수 선택)을 선택한 후, Lambda 함수 생성에서 생성한 Lambda 함수를 선택합니다.
8. 다음을 선택합니다. Function Configuration(함수 구성) 대화 상자가 표시됩니다.
9. IAM Role(IAM 역할)에서, Lambda 함수 생성에서 생성한 IAM 역할을 선택합니다.
10. 완료를 선택하면 Lambda 함수가 AWS에 업로드됩니다.

Lambda 함수 테스트

다음 AWS CLI 명령을 사용하여 비디오의 레이블 감지 분석을 시작하여 Lambda 함수를 테스트합니다. 분석을 완료하면 Lambda 함수가 트리거됩니다. CloudWatch 로그를 확인하여 분석이 성공했는지 확인합니다.

Lambda 함수를 테스트하려면

1. MOV 또는 MPEG-4 형식 비디오 파일을 S3 버킷으로 업로드합니다. 테스트할 때는 30초 이하의 비디오를 업로드합니다.

   이에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 Amazon S3에 객체 업로드를 참조하세요.

2. 다음 AWS CLI 명령을 실행하여 비디오에서 레이블 감지를 시작합니다.

```
aws rekognition start-label-detection --video
 "S3Object={Bucket="bucketname",Name="videofile"}" \
--notification-channel "SNSTopicArn=TopicARN,RoleArn=RoleARN" \
--region Region
```

다음 값을 업데이트합니다.

- bucketname 및 videofile을 Amazon S3 버킷 이름과 레이블을 감지하려는 비디오의 파일 이름으로 변경합니다.
- TopicARN을 SNS 주제 생성에서 생성한 Amazon SNS 주제의 ARN으로 바꿉니다.
- RoleARN을 SNS 주제 생성에서 생성한 IAM 역할의 ARN으로 바꿉니다.
- 사용 중인 AWS 리전Region으로 변경합니다.

3. 응답의 JobId 값을 기록합니다. 이 응답은 다음 JSON 예제와 비슷해 보입니다.

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

4. https://console.aws.amazon.com/cloudwatch/ 콘솔을 엽니다.
5. 분석이 완료되면 Lambda 함수의 로그 항목이 로그 그룹에 표시됩니다.
6. Lambda 함수를 선택하여 로그 스트림을 봅니다.
7. 최신 로그 스트림을 선택하여 Lambda 함수가 만든 로그 항목을 봅니다. 작업이 성공한 경우 다음 출력과 유사하게 표시되며, 작업 ID, 작업 유형 'StartLabelDetection' 및 Bottle, Clothing, Crowd, Food와 같은 감지된 레이블 범주 목록을 포함하여 비디오 인식 작업에 대한 세부 정보가 표시됩니다.

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:

작업 id 값은 3단계에서 기록해 둔 `JobId`의 값과 일치해야 합니다.

신원 확인을 위한 Amazon Rekognition 사용

Amazon Rekognition은 사용자에게 신원 확인 시스템을 간단하게 생성할 수 있는 여러 작업을 제공합니다. Amazon Rekognition을 사용하면 이미지에서 얼굴을 감지한 다음 얼굴 데이터를 비교하여 감지된 얼굴을 다른 얼굴과 비교할 수 있습니다. 이 얼굴 데이터는 컬렉션이라 불리는 서버 측 컨테이너에 저장됩니다. Amazon Rekognition의 얼굴 감지, 얼굴 비교 및 컬렉션 관리 작업을 활용하면 신원 확인 솔루션이 포함된 애플리케이션을 만들 수 있습니다.

이 자습서에서는 신원 확인이 필요한 애플리케이션을 생성하기 위한 두 가지 일반적인 워크플로를 보여줍니다.

첫 번째 워크플로는 컬렉션에 새 사용자를 등록하는 것입니다. 두 번째 워크플로는 재방문하는 사용자를 기록하기 위해 기존 컬렉션을 검색하는 것입니다.

이 자습서에서는 AWS SDK for Python을 사용합니다. 더 많은 Python 자습서를 보려면 AWS설명서 SDK 예제 GitHub 리포지토리를 참조하세요.

주제

- 사전 조건
- 모음 만들기
- 신규 사용자 등록
- 기존 사용자 로그인

사전 조건

이 자습서를 시작하기 전에 Python을 설치하고 Python AWSSDK를 설정하는 데 필요한 단계를 완료해야 합니다. 이 외에도 다음 사항을 반드시 갖추어야 합니다.

- AWS 계정 및 IAM 역할 생성
- Python SDK(Boto3) 설치
- AWS액세스 자격 증명을 올바르게 구성했습니다.
- Amazon Simple Storage Service 버킷을 생성하고 신원 확인을 위해 ID로 사용하려는 이미지를 업로드 완료함.

- 신원 확인을 위한 대상 이미지로 사용할 두 번째 이미지를 선택함.

모음 만들기

컬렉션에 새 사용자를 등록하거나 컬렉션에서 사용자를 검색하려면 먼저 사용할 컬렉션이 있어야 합니다. Amazon Rekognition 컬렉션은 감지된 얼굴에 대한 정보를 저장하는 데 사용되는 서버 측 컨테이너입니다.

컬렉션 생성

먼저 애플리케이션에서 사용할 컬렉션을 생성하는 함수를 작성해 보겠습니다. Amazon Rekognition 은 감지된 얼굴에 대한 정보를 컬렉션이라고 하는 서버 측 컨테이너에 저장합니다. 컬렉션에 저장된 얼굴 정보를 검색하여 알려진 얼굴을 찾을 수 있습니다. 얼굴 정보를 저장하려면 먼저 `CreateCollection` 작업을 사용하여 컬렉션을 만들어야 합니다.

1. 생성하려는 컬렉션의 이름을 선택합니다. 다음 코드에서 `collection_id`의 값을 만들려는 컬렉션의 이름으로 바꾸고, `region`의 값을 사용자 보안 인증 정보에 정의된 리전 이름으로 바꿉니다. 필수 사항은 아니지만 `Tags` 인수를 사용하여 컬렉션에 원하는 태그를 적용할 수 있습니다. 이 `CreateCollection` 작업을 수행하면 컬렉션의 ARN을 포함하여 생성한 컬렉션에 대한 정보가 반환됩니다. 코드를 실행한 결과로 받게 되는 ARN을 기록해 두세요.

```
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. 코드를 저장하고 실행합니다. 컬렉션 ARN을 복사하세요.

이제 Rekognition 컬렉션이 생성되었으므로 해당 컬렉션에 얼굴 정보와 식별자를 저장할 수 있습니다. 또한 검증을 위해 저장된 정보와 얼굴을 비교할 수 있습니다.

신규 사용자 등록

새 사용자를 등록하고 컬렉션에 사용자 정보를 추가할 수 있어야 합니다. 새 사용자를 등록하는 프로세스에는 일반적으로 다음 단계가 포함됩니다.

`DetectFaces` 작업 직접 호출

`DetectFaces` 작업을 통해 얼굴 이미지의 품질을 확인하는 코드를 작성하세요. 이 `DetectFaces` 작업을 통해 카메라로 찍은 이미지가 `SearchFacesByImage` 작업 처리에 적합한지 판단할 수 있습니다. 이미지에는 얼굴이 한 개만 포함되어야 합니다. `DetectFaces` 작업에 로컬 입력 이미지 파일을 제공하고 이미지에서 감지된 얼굴에 대한 세부 정보를 받게 됩니다. 다음 샘플 코드는 입력 이미지를 `DetectFaces`에 제공한 다음 이미지에서 얼굴이 하나만 감지되었는지 확인합니다.

1. 다음 코드 예제에서는 얼굴을 감지하려는 대상 이미지의 이름으로 `photo`를 바꿉니다. 또한 `region`의 값을 귀하의 계정과 연결된 리전의 이름으로 바꾸세요.

```
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. 진행 코드를 저장하고 실행합니다.

`CompareFaces` 작업 직접 호출

애플리케이션은 컬렉션에 새 사용자를 등록하고 재방문 사용자의 신원을 확인할 수 있어야 합니다. 새 사용자를 등록하는 데 사용되는 함수를 먼저 생성해야 합니다. 먼저 `CompareFaces` 작업을 사용하여 사용자의 로컬 입력 이미지 혹은 대상 이미지와 ID 또는 저장된 이미지를 비교합니다. 두 이미지에서 감지된 얼굴이 일치하는 경우 컬렉션을 검색하여 사용자가 컬렉션에 등록되어 있는지 확인할 수 있습니다.

먼저 입력 이미지를 Amazon S3 버킷에 저장한 ID 이미지와 비교하는 함수를 작성합니다. 다음 코드 예제에서는 입력 이미지를 직접 제공해야 합니다. 입력 이미지는 일종의 생체 확인 감지기를 사용한 후에 촬영해야 합니다. Amazon S3 버킷에 저장되어 있는 이미지의 이름도 전달해야 합니다.

1. `bucket`의 값을 소스 파일이 들어있는 Amazon S3 버킷의 이름으로 바꿉니다. 또한 `source_file`의 값을 사용 중인 소스 이미지의 이름으로 바꿔야 합니다. `target_file`의 값을 입력한 대상 파일의 이름으로 바꾸세요. `region`의 값을 사용자 보안 인증 정보에 정의된 `region` 이름으로 바꾸세요.

   또한 `similarityThreshold` 인수를 사용하여 응답으로 반환되기를 원하는 일치의 최소 신뢰도를 지정해야 합니다. 신뢰도가 이 임계값을 초과하는 경우에만 탐지된 얼굴이 `FaceMatches` 배열에 포함되어 반환됩니다. 선택한 `similarityThreshold`는 구체적 사용 사례의 특성을 반영해

야 합니다. 중대한 보안 애플리케이션과 관련된 모든 사용 사례는 임계값으로 99를 선택해야 합니다.

```
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. 진행 코드를 저장하고 실행합니다.

일치하는 얼굴에 대한 정보 및 신뢰도가 포함된 응답 개체가 반환됩니다.

SearchFacesByImage 작업 직접 호출

CompareFaces 작업의 신뢰도가 선택한 SimilarityThreshold보다 높으면 컬렉션에서 입력 이미지와 일치할 수 있는 얼굴을 검색해 보는 것이 좋습니다. 컬렉션에서 일치하는 항목이 발견되면 해당 사용자는 이미 컬렉션에 등록되어 있을 가능성이 높으므로 컬렉션에 새 사용자를 등록할 필요가 없습니다. 일치하는 사용자가 없는 경우 컬렉션에 새 사용자를 등록할 수 있습니다.

1. 먼저 SearchFacesByImage 작업을 간접적으로 호출할 코드를 작성하세요. 이 작업은 로컬 이미지 파일을 인수로 받아들인 다음 제공된 이미지에서 감지된 가장 큰 얼굴과 일치하는 얼굴을 Collection에서 검색합니다.

   다음 코드 예제에서 collectionId의 값을 검색하려는 컬렉션으로 변경합니다. region의 값을 귀하의 계정과 연결된 리전의 이름으로 바꾸세요. 또한 photo의 값을 입력 파일의 이름으로 바꿔야 합니다. threshold 값을 선택한 백분위수로 바꾸어 유사성 임계값도 지정해야 합니다.

```
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. 진행 코드를 저장하고 실행합니다. 일치하는 항목이 있으면 이미지에서 인식된 사람이 이미 컬렉션에 속해 있다는 의미이므로 다음 단계를 진행할 필요가 없습니다. 이 경우에는 애플리케이션에 대한 사용자 액세스를 허용하기만 하면 됩니다.

IndexFaces 작업 직접 호출

검색한 컬렉션에 일치하는 항목이 없었다면 컬렉션에 사용자의 얼굴을 추가해야 합니다. IndexFaces 작업을 직접 호출해서 이를 수행할 수 있습니다. IndexFaces를 직접 호출하면 Amazon Rekognition은 입력 이미지에서 식별된 얼굴의 특성을 추출하여 지정된 컬렉션에 데이터를 저장합니다.

1. 먼저 IndexFaces를 직접 호출할 코드를 작성합니다. image의 값을 IndexFaces 작업의 입력 이미지로 사용할 로컬 파일 이름으로 바꿉니다. 또한 photo_name의 값을 원하는 입력 파일의 이름으로 바꿔야 합니다. collection_id의 값을 이전에 만든 컬렉션의 ID로 바꾸는 것도 잊지 마세요. 다음으로 region의 값을 귀하의 계정과 연결된 리전의 이름으로 바꾸세요. 또한 인덱싱해야 하는 이미지의 최대 얼굴 수를 정의하는 MaxFaces 입력 파라미터 값을 지정하는 것이 좋습니다. 이 파라미터의 기본값은 1입니다.

```
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. 진행 코드를 저장하고 실행합니다. 이미지 속 인물에게 할당된 FaceID와 같은 `IndexFaces` 작업에서 반환된 데이터를 저장할지 결정하세요. 다음 섹션에서는 이 데이터를 저장하는 방법을 살펴보겠습니다. 계속하기 전에 반환된`FaceId`, `ImageId`, 및 `Confidence` 값을 복사해 두세요.

Amazon S3와 Amazon DynamoDB에 이미지 및 FaceID 데이터 저장

입력 이미지의 Face ID를 가져오면 이미지 데이터를 Amazon S3에 저장하고, 얼굴 데이터와 이미지 URL은 DynamoDB와 같은 데이터베이스에 입력할 수 있습니다.

1. 입력 이미지를 Amazon S3 데이터베이스에 업로드하는 코드를 작성합니다. 다음 코드 샘플에서 `bucket` 값을 파일을 업로드하려는 버킷의 이름으로 대체한 다음, `file_name` 값을 Amazon S3 버킷에 저장하려는 로컬 파일의 이름으로 대체합니다. `key_name` 값을 이미지 파일에 지정하고자 하는 이름으로 대체하여 Amazon S3 버킷에 있는 파일을 식별할 키 이름을 입력합니다. 업로드하려는 파일은 이전 코드 샘플에서 정의한 파일, 즉 IndexFaces에 사용한 입력 파일과 동일합니다. 마지막으로 `region`의 값을 귀하의 계정과 연결된 리전의 이름으로 바꾸세요.

```
import boto3
import logging
from botocore.exceptions import ClientError
```

```
# store local file in S3 bucket
bucket = "amzn-s3-demo-bucket"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
s3 = boto3.client('s3', region_name=region)
# Upload the file
try:
    response = s3.upload_file(file_name, bucket, key_name)
    print("File upload successful!")
except ClientError as e:
    logging.error(e)
```

2. 진행 코드 샘플을 저장하고 실행하여 입력 이미지를 Amazon S3에 업로드합니다.
3. 반환된 Face ID도 데이터베이스에 저장해야 할 것입니다. DynamoDB 데이터베이스 테이블을 생성한 다음 Face ID를 해당 테이블에 업로드하면 됩니다. 다음 코드 샘플에서는 DynamoDB 테이블을 생성합니다. 단, 이 테이블을 생성하는 코드는 한 번만 실행하면 된다는 것을 알아 두세요. 다음 코드에서 `region`의 값을 귀하의 계정과 연결된 리전의 이름으로 바꾸세요. 또한 `database_name` 값을 DynamoDB 테이블에 지정하려는 이름으로 바꿔야 합니다.

```
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. 진행 코드를 저장하고 실행하여 테이블을 생성합니다.
5. 테이블을 생성한 후 반환된 FaceId를 테이블에 업로드할 수 있습니다. 이를 위해 Table 함수를 사용하여 테이블로의 연결을 설정한 다음 put_item 함수를 사용하여 데이터를 업로드해야 합니다.

   다음 코드 샘플에서 bucket의 값을 Amazon S3에 업로드한 입력 이미지가 들어 있는 버킷 이름으로 대체합니다. 또한 file_name의 값을 Amazon S3 버킷에 업로드한 입력 파일의 이름으로 바꾸고 key_name의 값을 이전에 입력 파일을 식별하는 데 사용한 키로 바꿔야 합니다. 마지막으로 region의 값을 귀하의 계정과 연결된 리전의 이름으로 바꾸세요. 이 값은 1단계에서 제공한 값과 일치해야 합니다.

   AddDBEntry는 얼굴에 할당된 FaceId, ImageId 및 신뢰도 값을 컬렉션에 저장합니다. IndexFaces 섹션의 2단계에서 저장한 값을 아래 함수에 제공하세요.

```
import boto3
from pprint import pprint
from decimal import Decimal
import json

# The local file that was stored in S3 bucket
bucket = "amzn-s3-demo-bucket"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
# Get URL of file
file_url = "https://s3.amazonaws.com/{}/{}".format(bucket, key_name)
print(file_url)

# upload face-id, face info, and image url
def AddDBEntry(file_name, file_url, face_id, image_id, confidence):
    dynamodb = boto3.resource('dynamodb', region_name=region)
    table = dynamodb.Table('FacesDB-4')
    response = table.put_item(
       Item={
            'ExternalImageID': file_name,
            'ImageURL': file_url,
            'FaceID': face_id,
            'ImageID': image_id,
            'Confidence': json.loads(json.dumps(confidence), parse_float=Decimal)
```

```
        }
    )
    return response

# Mock values for face ID, image ID, and confidence - replace them with actual
 values from your collection results
dynamodb_resp = AddDBEntry(file_name, file_url, "FACE-ID-HERE",
    "IMAGE-ID-HERE", confidence-here)
print("Database entry successful.")
pprint(dynamodb_resp, sort_dicts=False)
```

6. 진행 코드 샘플을 저장하고 실행하여 반환된 Face ID 데이터를 테이블에 저장합니다.

기존 사용자 로그인

컬렉션에 사용자를 등록한 후에는 해당 사용자가 다시 돌아오면 SearchFacesByImage 작업을 사용하여 인증을 받을 수 있습니다. 입력 이미지를 가져온 다음 DetectFaces를 사용하여 입력 이미지의 품질을 확인해야 합니다. 이를 통해 SearchFacesbyImage 작업을 실행하기 전에 적절한 이미지를 사용했는지 확인할 수 있습니다.

DetectFaces 작업 직접 호출

1. DetectFaces 작업을 통해 얼굴 이미지의 품질을 확인하고 카메라로 찍은 이미지가 SearchFacesByImage 작업 처리에 적합한지 판단할 수 있습니다. 입력 이미지에는 얼굴이 한 개만 포함되어야 합니다. 다음 코드 샘플은 입력 이미지를 가져와 DetectFaces 작업에 제공합니다.

   다음 코드 예제에서는 photo 값을 로컬 대상 이미지의 이름으로 바꾸고 region 값을 귀하의 계정과 연결된 리전 이름으로 바꿉니다.

```
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. 코드를 저장하고 실행합니다.

SearchFacesByImage 작업 직접 호출

1. 감지된 얼굴을 SearchFacesByImage를 사용하여 컬렉션에 있는 얼굴과 비교하는 코드를 작성합니다. 신규 사용자 등록 섹션에 표시된 코드를 사용하여 SearchFacesByImage 작업에 입력 이미지를 제공합니다.

   다음 코드 예제에서 collectionId의 값을 검색하려는 컬렉션으로 변경합니다. 또한 bucket 값을 Amazon S3 버킷의 이름으로 변경하고 fileName의 값을 해당 버킷에 있는 이미지 파일로 변경합니다. region의 값을 귀하의 계정과 연결된 리전의 이름으로 바꾸세요. threshold 값을 선택한 백분위수로 바꾸어 유사성 임계값도 지정해야 합니다.

```
import boto3
```

```
bucket = 'amzn-s3-demo-bucket'
collectionId = 'collection-id-name'
region = "region-name"
fileName = 'file-name'
threshold = 70
maxFaces = 1
client = boto3.client('rekognition', region_name=region)

# input image should be local file here, not s3 file
with open(fileName, 'rb') as image:
    response = client.search_faces_by_image(CollectionId=collectionId,
    Image={'Bytes': image.read()},
    FaceMatchThreshold=threshold, MaxFaces=maxFaces)
```

2. 코드를 저장하고 실행합니다.

반환된 FaceID 및 신뢰도 수준 확인

이제 FaceId, 유사성, 신뢰도 속성과 같은 응답 요소를 출력하여 일치하는 FaceId에 대한 정보를 확인할 수 있습니다.

```
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']))
```

Amazon Rekognition(REK)을 사용하여 마케팅 애플리케이션의 레이블 감지

이 자습서에서는 웹 사이트에 업로드된 이미지를 기반으로 사람들에게 이메일을 보내는 데 사용할 수 있는 샘플 Python 애플리케이션을 구축하는 방법을 안내합니다. 샘플 애플리케이션은 사진에서 랜드마크가 인식되는 경우 여행 상품에 대한 개인 맞춤형 이메일을 전송하여 소셜 미디어 마케팅 캠페인에 사용자를 참여시키도록 설계되었습니다.

이 솔루션은 Amazon Rekognition, Amazon S3, DynamoDB, CloudWatch 및 Amazon SES를 비롯한 다양한 AWS 서비스를 통합합니다. 애플리케이션은 Amazon Rekognition을 사용하여 DynamoDB 데

이터베이스에서 인식된 계정이 업로드한 이미지에서 레이블을 감지한 다음, 이미지에서 감지된 레이블을 기반으로 계정 소유자에게 마케팅 이메일을 전송합니다. 전체 솔루션 아키텍처는 다음과 같습니다.

- DynamoDB 데이터베이스에 사용자 데이터를 저장합니다.
- 사용자는 이미지 데이터 및 메타데이터(사용자 계정 번호)를 Amazon S3에 업로드합니다.
- Lambda 함수는 DetectLabels를 호출하여 업로드된 이미지에서 레이블을 식별 및 기록하고, DynamoDB에서 사용자 이메일을 조회하며, 이미지를 업로드한 사용자에게 마케팅 이메일을 보냅니다.
- Lambda 함수의 결과는 나중에 검토할 수 있도록 CloudWatch에 기록됩니다.

다음은 자습서의 모든 단계에 대한 개요입니다.

1. DynamoDB 테이블 생성 및 채우기
2. 로깅 및 알림을 사용하여 Lambda 함수를 작성합니다.
3. Lambda 함수 및 권한을 설정합니다.
4. Amazon S3를 사용하여 이미지 및 메타데이터를 업로드합니다.
5. CloudWatch 로그를 폴링합니다.

주제

- 사전 조건
- DynamoDB 테이블 생성 및 채우기
- 로깅 및 알림을 사용하여 Lambda 함수 생성
- 권한 및 Lambda 함수 설정
- 메타데이터를 사용하여 Amazon S3에 이미지 업로드
- 클라이언트에서 CloudWatch 로그 폴링

사전 조건

이 자습서를 시작하려면 다음이 필요합니다.

- AWS 계정 및 적절한 IAM 권한.
- 개발 환경에 Python 및 Boto3 설치.
- Lambda, Amazon S3, DynamoDB, Amazon SES, CloudWatch에 대한 기본 이해.

- Lambda 및 Amazon SES에 대한 기본적인 지식.

DynamoDB 테이블 생성 및 채우기

이 자습서를 시작하려면 프라이머리 키로 ‘AccountNumber’ 변수를 사용하여 이메일 주소, 연령, 전화 번호 및 멤버십 상태와 같은 고객 데이터를 저장하는 DynamoDB 테이블을 생성합니다. 또한 일부 샘플 데이터를 테이블에 삽입합니다.

DynamoDB 테이블 생성

먼저 DynamoDB 테이블을 생성합니다. Python 및 Boto3를 사용하여 설정하는 방법은 다음과 같습니다. `create_user_table`이라는 함수를 생성하고 그 안에 있는 DynamoDB 리소스에 연결합니다. 아래 코드 샘플에서 'region_name' 값을 계정이 작동하는 리전의 이름으로 바꾼 다음 코드 셀을 실행하여 테이블을 생성합니다.

```
import boto3
def create_user_table(dynamodb=None):
    if not dynamodb:
        dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

    table = dynamodb.create_table(
        TableName='CustomerDataTable',
        KeySchema=[
            {
                'AttributeName': 'AccountNumber',
                'KeyType': 'HASH'  # Partition key
            },
        ],
        AttributeDefinitions=[
            {
                'AttributeName': 'AccountNumber',
                'AttributeType': 'S'
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        }
    )

    # Wait until the table exists.
```

```
    table.wait_until_exists()
    print("Table status:", table.table_status)

# Create the DynamoDB table.
create_user_table()
```

이 스크립트를 실행하면 AccountNumber이 있는 CustomerDataTable이라는 DynamoDB 테이블을 프라이머리 키로 설정합니다.

샘플 데이터 삽입

이제 일부 샘플 데이터를 테이블에 삽입하겠습니다. 이 샘플 데이터는 애플리케이션의 전체 기능을 테스트하는 데 도움이 됩니다.

이전에 생성한 것과 동일한 CustomerDataTable에 샘플 데이터를 추가하는 두 번째 함수를 생성합니다. 아래 코드로 3개의 샘플 항목이 생성되며 각 항목에는 계정 번호, 이메일 주소, 연령, 전화번호 및 멤버십 상태가 포함됩니다. 아래 코드 샘플에서 region_name 값을 계정이 작동하는 리전의 이름으로 바꾼 다음 코드 셀을 실행하여 테이블을 생성합니다. 앱의 이메일 전송 부분을 테스트하려면 아래 첫 번째 고객 항목의 EmailAddress 값을 이메일을 받을 수 있는 이메일 주소로 바꿉니다. 코드를 저장하고 실행합니다.

```
import boto3

def insert_sample_data(dynamodb=None):
    if not dynamodb:
        dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

    table = dynamodb.Table('CustomerDataTable')

    # Sample data
    customers = [
        {
            'AccountNumber': 'ACC1000',
            'EmailAddress': 'email-for-delivery-here',
            'Age': 30,
            'PhoneNumber': '123-456-7890',
            'MembershipStatus': 'Active'
        },
        {
            'AccountNumber': 'ACC1001',
```

```
            'EmailAddress': 'jane.doe@example.com',
            'Age': 25,
            'PhoneNumber': '098-765-4321',
            'MembershipStatus': 'Inactive'
        },
        {
            'AccountNumber': 'ACC1002',
            'EmailAddress': 'pat.candella@example.com',
            'Age': 35,
            'PhoneNumber': '555-555-5555',
            'MembershipStatus': 'Active'
        }
    ]
        # Inserting data
    for customer in customers:
            print(f"Adding customer: {customer['AccountNumber']}")
            table.put_item(Item=customer)

# Insert sample data into DynamoDB
insert_sample_data()
```

DynamoDB 테이블을 설정하고 채우면 이제 이 데이터 검색을 Lambda 함수에 통합할 수 있습니다. 이를 통해 애플리케이션은 향후 이미지 처리 워크플로에서 식별할 계정 번호를 기반으로 사용자 세부 정보를 검색할 수 있습니다.

로깅 및 알림을 사용하여 Lambda 함수 생성

이제 Lambda 함수를 생성할 수 있습니다. 이미지 업로드 시 트리거되는 Lambda 함수가 이미지 데이터와 계정 메타데이터를 모두 읽고 이를 사용하여 DynamoDB 테이블에서 관련 사용자 데이터를 조회할 수 있는지 확인하고자 합니다. 즉, Amazon Rekognition의 DetectLabels 함수를 호출해야 할 뿐만 아니라 AccountNumber를 가져와 DynamoDB에서 연결된 이메일 주소를 검색하는 데 사용하는 함수도 필요합니다. 또한 Amazon SES를 사용하여 이메일 주소로 이메일을 보내는 함수가 필요합니다. 마지막으로 로거를 사용하여 프로세스에 대한 정보를 기록합니다. 이 데이터는 CloudWatch에 표시됩니다.

Lambda 함수 생성

다음은 Lambda 함수가 이러한 요구 사항을 처리하는 방법에 대한 개요입니다. 다음 코드 샘플에서 CustomerDataTable 이외의 다른 것을 테이블 이름으로 사용한 경우 올바른 DynamoDB 테이블이 지정되어 있는지 확인합니다. 또한 'send_marketing_email' 함수에서 'Source' 인수의 값을 전송 이메일 역할을 하는 액세스 권한이 있는 이메일 주소로 바꿔야 합니다.

```
import json
import boto3
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    s3_bucket = event['Records'][0]['s3']['bucket']['name']
    s3_object_key = event['Records'][0]['s3']['object']['key']
    print(s3_bucket)
    print(s3_object_key)

    s3 = boto3.client('s3')
    try:
        s3_response = s3.head_object(Bucket=s3_bucket, Key=s3_object_key)
        account_number = s3_response['Metadata']['account_number']
    except Exception as e:
        logger.error(f"Failed to retrieve object or metadata: {str(e)}")
        raise e  # Optionally re-raise to handle the error upstream or signal a failure

    rekognition = boto3.client('rekognition')
    try:
        labels_response = rekognition.detect_labels(Image={'S3Object': {'Bucket':
 s3_bucket, 'Name': s3_object_key}})
        #logger.info(f"Detected labels: {json.dumps(labels_response['Labels'],
 indent=4)}")
    except Exception as e:
        #logger.info(f"Detected label: {label['Name']}")
        raise e

    #logger.info(f"Detected labels: {json.dumps(labels_response['Labels'], indent=4)}")

    landmark_detected = any(label['Name'] == 'Landmark' and label['Confidence'] > 20
 for label in labels_response['Labels'])
    if landmark_detected:
        result = notify_user_based_on_landmark(account_number)
        print(result)
        #logger.info(f"Detected label: {label['Name']}")
        #logger.info(f"Notification sent: {result}")
    return {
        'statusCode': 200,
        'body': json.dumps('Process completed successfully!')
```

```
    }

def notify_user_based_on_landmark(account_number):
     # Retrieve user data from DynamoDB
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('CustomerDataTable')
    user_info = table.get_item(Key={'AccountNumber': account_number})

    # Send email if user is found
    if 'Item' in user_info:
        send_marketing_email(user_info['Item']['EmailAddress'])
    return "User notified with relevant travel information."

def send_marketing_email(email):
    ses = boto3.client('ses')
    response = ses.send_email(
        Source='your-email@example.com',
        Destination={'ToAddresses': [email]},
        Message={
            'Subject': {'Data': 'Explore New Destinations!'},
            'Body': {
                'Text': {'Data': 'Check out our exclusive travel packages inspired by
 the landmark in your uploaded image!'}
            }
        }
    )
    return f"Email sent to {email} with status {response['ResponseMetadata']
['HTTPStatusCode']}"
    print("succeess")
```

이제 Lambda 함수를 작성했으므로 Lambda에 대한 권한을 구성하고 AWS 관리 콘솔에서 Lambda 함수의 인스턴스를 생성해야 합니다.

권한 및 Lambda 함수 설정

IAM 역할 생성 또는 업데이트

Lambda 함수를 사용하여 사용자가 업로드한 이미지를 처리하려면 먼저 해당 권한을 처리해야 합니다. Lambda 함수에는 Amazon S3, DynamoDB, Amazon SES, Amazon Rekognition 및 CloudWatch와 상호 작용할 수 있는 정책이 있는 범위가 지정된 IAM 역할이 필요합니다.

Lambda 함수용 IAM 역할을 구성하려면

1. AWS Management Console을 엽니다.
2. IAM > 역할 > 역할 생성으로 이동합니다.

   ‘역할 생성’을 선택합니다.
3. 이 역할을 사용할 서비스로 Lambda를 선택합니다. ‘다음’을 클릭합니다.
4. 다음 정책을 연결합니다(이러한 정책은 실제 프로덕션 설정에서 필요한 정책으로만 권한을 제한하려는 데모용으로만 선택됩니다).

   - AmazonS3ReadOnlyAccess
   - AmazonRekognitionReadOnlyAccess
   - AmazonDynamoDBFullAccess
   - AmazonSESFullAccess
   - AWSLambdaExecute
   - AWSLambdaBasicExecutionRole(CloudWatch 로깅용)

   ‘다음’을 클릭합니다.
5. 역할의 이름과 설명을 지정한 다음 '역할 생성'을 선택하여 역할을 생성합니다.
6. Lambda에 대한 적절한 권한을 구성한 후에는 AWS 관리 콘솔을 사용하여 Lambda 함수의 인스턴스를 생성할 수 있습니다.

   AWS Management Console에서 Lambda 서비스로 이동합니다.
7. ‘함수 생성’을 클릭합니다. ‘새로 작성’을 선택합니다.
8. 함수 이름을 입력하고 Python 런타임을 선택합니다. '기본 실행 역할 변경'에서 '기존 역할 사용'을 선택한 다음 이전에 생성한 IAM 역할을 선택합니다.
9. 이제 lambda 함수 탭에서 코드를 생성하고 업데이트해야 합니다. ‘lambda_function’이라는 탭으로 이동하여 해당 코드를 진행 중인 Lambda 코드 샘플로 바꿉니다.

   변경 사항을 저장하고 배포합니다.
10. 이제 Amazon S3 이벤트를 Lambda 트리거로 구성해야 합니다.

    Lambda 함수의 구성 탭/페이지에서 트리거로 이동하여 ‘트리거 추가’를 클릭합니다.
11. 사용 가능한 트리거 목록에서 Amazon S3를 선택합니다.
12. 트리거를 구성하려면 함수를 트리거할 버킷을 선택합니다.

이벤트 유형 PUT를 선택합니다. 선택적으로 특정 이름 또는 유형의 파일만 처리하려는 경우 접두사 또는 접미사를 지정할 수 있습니다.

13. ‘추가’를 클릭하여 트리거를 활성화하고 구성을 저장합니다.

Amazon SES 이메일 주소 확인

Amazon SES를 사용하여 이메일을 전송하려면 먼저 발신자와 수신자 모두의 이메일 주소를 확인해야 합니다. 방법:

1. Amazon SES 콘솔로 이동합니다. 'ID 관리'로 이동한 다음 '이메일 주소'로 이동합니다.
2. ‘새 이메일 주소 확인’을 클릭합니다. 확인하려는 이메일 주소를 입력하고 ‘이 이메일 주소 확인’을 클릭합니다. 확인 링크가 포함된 이메일을 받게 됩니다. 링크를 클릭하여 확인 프로세스를 완료합니다.
3. 두 계정의 이메일 주소를 확인하고, Lambda 함수를 올바른 권한으로 설정한 다음, 일부 샘플 고객 데이터를 생성한 후 선택한 Amazon S3 버킷에 테스트 이미지를 업로드하여 Lambda 함수를 테스트할 수 있습니다.

메타데이터를 사용하여 Amazon S3에 이미지 업로드

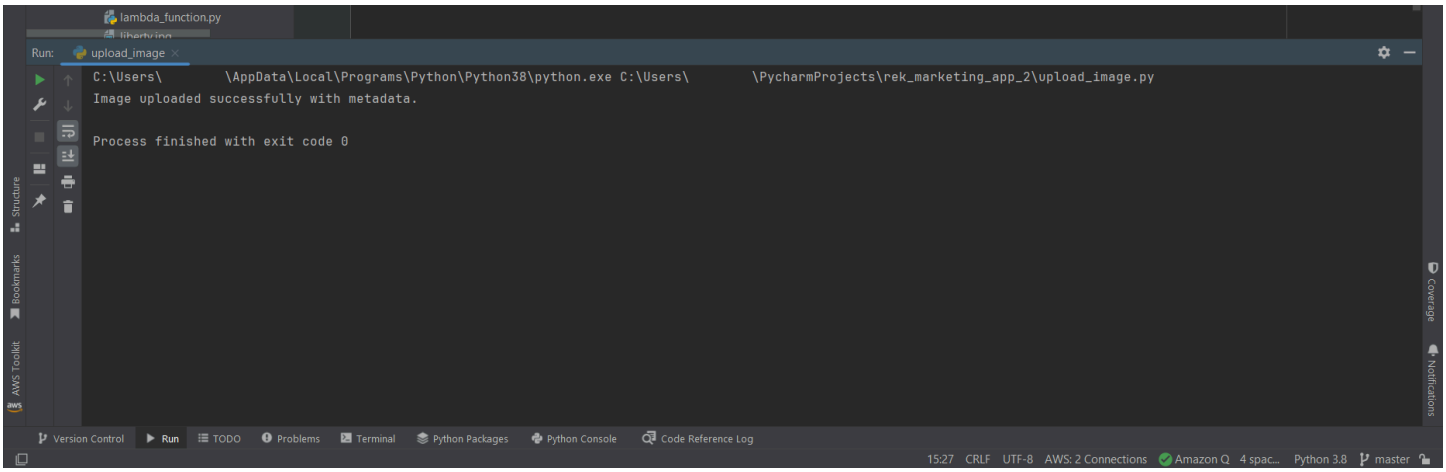
이제 AWS 콘솔 또는 앞서 준비한 스크립트를 사용하여 선택한 Amazon S3 버킷에 테스트 이미지를 업로드할 수 있습니다. Lambda 함수에서 이전에 지정한 Amazon S3 버킷에 이미지를 업로드하는 스크립트를 작성합니다. 아래 코드 샘플에서 이미지의 경로를 'upload_image_to_s3'에 대한 첫 번째 인수로 지정하고, 버킷 이름을 두 번째 인수로 지정하며, 이미지를 업로드하는 사용자의 계정 번호를 최종 인수로 지정합니다.

```
import boto3
def upload_image_to_s3(file_name, bucket, account_number):
    s3 = boto3.client('s3')
    try:
        with open(file_name, 'rb') as data:
            s3.upload_fileobj(
                Fileobj=data,
                Bucket=bucket,
                Key=file_name,
                ExtraArgs={
                    'Metadata': {'account_number': account_number}
                }
            )
```

```
        print("Image uploaded successfully with metadata.")
    except Exception as e:
        print("Failed to upload image")
        print(e)

# Usage
upload_image_to_s3('path-to-image-here', 's3-bucket-name-here', 'user-account-number-
here')
```

이 함수에서는 upload_fileobj 메서드의 ExtraArgs 파라미터가 이미지와 함께 사용자 정의 메타데이터(account_number)를 포함하는 데 사용됩니다. 나중에이 메타데이터 AWS 를 사용하여 그에 따라 이미지를 처리할 수 있습니다.

저장하고 스크립트를 실행합니다. 그러면 이미지가 업로드됩니다.



이미지를 업로드한 후 몇 분 후에 여기에 지정된 계정과 이전에 연결한 주소로 이메일을 받게 됩니다.

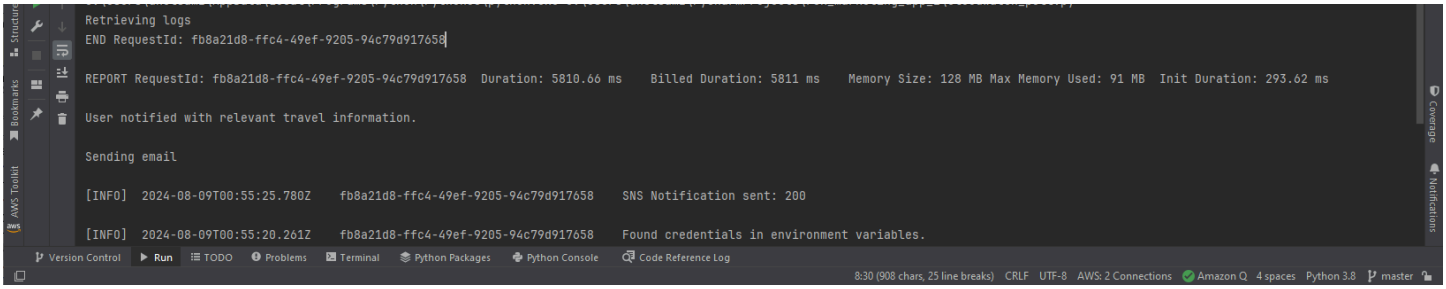
클라이언트에서 CloudWatch 로그 폴링

Lambda 함수의 CloudWatch 로그를 확인하여 예상대로 트리거되고 실행되었는지 확인합니다. 로그는 CloudWatch > 로그 > 로그 그룹 > /AWS/lambda/your_function_name에서 찾을 수 있습니다. 스크립트를 작성하여 로그에 액세스하고 인쇄할 수도 있습니다. 다음 코드 샘플은 Lambda 그룹에서 로그를 폴링하여 지난 한 시간 동안 생성된 로그를 출력합니다. 코드를 저장하고 실행합니다.

```
import boto3
import time

def fetch_lambda_logs(log_group_name, start_time):
    client = boto3.client('logs')
```

```
    query = "fields @timestamp, @message | sort @timestamp desc | limit 20"
    start_query_response = client.start_query(
        logGroupName=log_group_name,
        startTime=int(start_time),
        endTime=int(time.time()),
        queryString=query,
    )
    query_id = start_query_response['queryId']
    response = None
    while response is None or response['status'] == 'Running':
        time.sleep(1)  # Wait for 1 second before checking the query status again
        response = client.get_query_results(queryId=query_id)
    return response['results']

# Usage
log_group = '/aws/lambda/RekMediaFunction'
logs = fetch_lambda_logs(log_group, int(time.time()) - 3600)  # Fetch logs from the
 last hour
print("Retrieving logs")
for log in logs:
    #print(log)
    print(log[1]['value'])
```

코드를 실행하면 로그가 인쇄되고 관련 여행 정보가 포함된 이메일을 통해 사용자에게 알림이 전송되었는지 확인할 수 있습니다.



Amazon S3 버킷에 업로드된 이미지의 레이블을 감지한 다음 이미지를 업로드한 사용자에게 프로모션 메시지를 이메일로 보낼 수 있는 애플리케이션을 성공적으로 생성했습니다. 더 이상 필요하지 않은 리소스는 삭제하여 요금이 부과되지 않도록 하세요.

Lambda와 Python을 사용하여 이미지에서 레이블 감지

AWS Lambda 는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행하는 데 사용할 수 있는 컴퓨팅 서비스입니다. Lambda 함수 내에서 Rekognition API 작업을 직접 호출할 수 있습니다. 다음 지침은 `DetectLabels`를 직접 호출하는 Lambda 함수를 Python에서 생성하는 방법을 보여줍니다.

Lambda 함수가 `DetectLabels`를 직접 호출하면 이미지에서 감지된 레이블 배열과 해당 레이블 감지의 신뢰도 수준을 반환합니다.

이 지침에는 Amazon S3 버킷 또는 로컬 컴퓨터에서 가져온 이미지를 사용하여 Lambda 함수를 직접 호출하는 방법을 보여주는 샘플 Python 코드가 포함되어 있습니다.

선택한 이미지가 Rekognition의 제한을 충족하는지 확인하세요. 이미지 파일 유형 및 크기 제한에 대한 자세한 내용은 Rekognition의 지침 및 할당량 및 DetectLabels API 참조를 참조하세요.

Lambda 함수 생성(콘솔)

이 단계에서는 빈 Lambda 함수와 Lambda 함수가 `DetectLabels` 작업을 직접 호출하게 해 주는 IAM 실행 역할을 생성합니다. 나중 단계에서 Lambda 함수에 소스 코드와 계층(선택 사항)을 추가합니다.

Amazon S3 버킷에 저장된 문서를 사용하는 경우, 이 단계에서는 문서를 저장하는 버킷에 대한 액세스 권한을 부여하는 방법도 보여줍니다.

AWS Lambda 함수를 생성하려면(콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
2. 함수 생성을 선택합니다. 자세한 내용은 콘솔로 Lambda 함수 생성을 참조하세요.
3. 다음과 같은 옵션을 선택하세요.
   - 새로 작성을 선택합니다.
   - 함수 이름의 값을 입력합니다.
   - 런타임의 경우 최신 버전의 Python을 선택합니다.
   - 아키텍처에서는 x86_64를 선택합니다.
4. 함수 생성을 선택하여 AWS Lambda 함수를 생성합니다.
5. 함수 페이지에서 구성 탭을 선택합니다.
6. 권한 창의 실행 역할에서 IAM 콘솔의 해당 역할을 열 역할 이름을 선택합니다.
7. 권한 탭에서 권한 추가, 그 다음 인라인 정책 생성을 선택합니다.

8. JSON을 선택하고 정책을 다음 정책으로 바꿉니다.

   JSON

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

9. 정책 검토를 선택합니다.
10. 정책의 이름을 입력합니다(예: DetectLabels-access).
11. 정책 생성을 선택합니다.
12. Amazon S3 버킷에 분석용 문서를 저장하는 경우 반드시 Amazon S3 액세스 정책을 추가해야 합니다. 이를 위해서는 AWS Lambda 콘솔에서 7~11단계를 반복하고 다음과 같이 변경합니다.

    a. 8단계에서는 다음 정책을 사용하세요. *##/## ##*를 Amazon S3 버킷 및 분석하려는 문서의 폴더 경로로 바꿉니다.

       JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::bucket/folder path/*"
        }
    ]
}
```

    b. 10단계에서는 S3Bucket-access와 같은 다른 정책 이름을 선택합니다.

(선택 사항) 계층 생성(콘솔)

Lambda 함수 및 `DetectLabels` 직접 호출을 사용하기 위해 이 단계를 수행할 필요는 없습니다.

`DetectLabels` 작업은 AWS SDK for Python(Boto3)의 일부로 기본 Lambda Python 환경에 포함됩니다.

Lambda 함수의 다른 부분에서 기본 Lambda Python 환경에 없는 최신 AWS 서비스 업데이트가 필요한 경우이 단계를 수행하여 최신 Boto3 SDK 릴리스를 함수에 계층으로 추가할 수 있습니다.

해당 SDK를 계층으로 추가하려면 먼저 Boto3 SDK가 포함된 .zip 파일 아카이브를 생성합니다. 그런 다음 계층을 생성하고 .zip 파일 아카이브를 계층에 추가합니다. 자세한 내용은 Lambda 함수에서 계층 사용을 참조하세요.

계층을 생성하고 추가하려면(콘솔)

1. 명령 프롬프트를 열고 다음 명령을 입력하여 AWS SDK의 최신 버전으로 배포 패키지를 생성합니다.

   ```
   pip install boto3 --target python/.
   zip boto3-layer.zip -r python/
   ```

2. 이 절차의 8단계에서 사용하는 zip 파일(boto3-layer.zip)의 이름을 기록해 둡니다.
3. https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
4. 탐색 창에서 계층을 선택합니다.
5. 계층 생성을 선택합니다.
6. Name(이름)과 Description(설명)을 입력합니다.
7. 코드 항목 유형에서 .zip 파일 업로드를 선택하고 업로드를 선택합니다.
8. 대화 상자에서 이 절차의 1단계에서 만든 zip 파일 아카이브(boto3-layer.zip)를 선택합니다.
9. 호환 런타임의 경우 최신 버전의 Python을 선택하세요.
10. 생성을 선택하여 계층을 생성합니다.
11. 탐색 창 메뉴 아이콘을 선택합니다.
12. 탐색 창에서 함수를 선택합니다.
13. 리소스 목록에서 ???에서 생성한 함수를 선택합니다.
14. 코드 탭을 선택합니다.
15. 계층 영역에서 계층 추가를 선택합니다.

16. 사용자 지정 계층을 선택합니다.
17. 사용자 지정 계층에서 6단계에서 입력한 계층 이름을 선택합니다.
18. 버전에서 계층 버전을 선택합니다. 계층 버전은 1이어야 합니다.
19. 추가를 선택합니다.

Python 코드 추가(콘솔)

이 단계에서는 Lambda 콘솔 코드 편집기를 사용하여 Lambda 함수에 Python 코드를 추가합니다. 이 코드는 `DetectLabels` 작업을 사용하여 이미지에서 레이블을 감지합니다. 이미지에서 감지된 레이블 배열과 해당 레이블 감지의 신뢰도 수준이 반환됩니다.

`DetectLabels` 작업에 제공하는 문서는 Amazon S3 버킷 또는 로컬 컴퓨터에 위치할 수 있습니다.

Python 코드를 추가하려면(콘솔)

1. 코드 탭으로 이동합니다.
2. 코드 편집기에서 lambda_function.py의 코드를 다음과 같이 바꾸세요.

```
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. Lambda 함수를 배포하려면 배포를 선택합니다.

Python 코드를 추가하려면(콘솔)

이제 Lambda 함수를 생성했으므로 이를 간접적으로 호출하여 이미지에서 레이블을 감지할 수 있습니다.

이 단계에서는 컴퓨터의 Python 코드를 실행하여 로컬 이미지 또는 Amazon S3 버킷의 이미지를 Lambda 함수에 전달합니다.

Lambda 함수를 생성한 리전과 동일한 AWS 리전에서 코드를 실행해야 합니다. Lambda 콘솔의 함수 세부 정보 페이지의 탐색 모음에서 Lambda 함수의 AWS 리전을 볼 수 있습니다.

Lambda 함수가 타임아웃 오류를 반환하는 경우 Lambda 함수의 제한 시간을 연장하세요. 자세한 내용은 함수 제한 시간 구성(콘솔)을 참조하세요.

코드에서 Lambda 함수를 간접 호출하는 방법에 대한 자세한 내용은 AWS Lambda 함수 간접 호출을 참조하세요.

Lambda 함수를 테스트하려면

1. 아직 수행하지 않은 경우 다음 작업을 수행하세요.

   a. 사용자에게 `lambda:InvokeFunction` 권한이 있는지 확인하세요. 다음 정책을 사용할 수 있습니다.

      Lambda 콘솔의 함수 개요에서 Lambda 함수에 대한 ARN을 가져올 수 있습니다.

      액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요.

      - 의 사용자 및 그룹 AWS IAM Identity Center:

        권한 세트를 생성합니다. AWS IAM Identity Center 사용자 안내서에서 권한 세트 생성의 지침을 따릅니다.

- ID 제공업체를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용자 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용자 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따릅니다.

b. AWS SDK for Python을 설치하고 구성합니다. 자세한 내용은 2단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 코드를 `client.py`라는 이름의 파일에 저장합니다.

```
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. 코드를 실행합니다. 문서가 Amazon S3 버킷에 있는 경우, ???의 12단계에서 지정한 버킷과 동일한지 확인하세요.

   성공하면 코드는 문서에서 감지된 각 블록 유형에 대해 부분 JSON 응답을 반환합니다.

AWS SDKs를 사용한 Amazon Rekognition 코드 예제

다음 코드 예제에서는 Amazon Rekognition을 AWS 소프트웨어 개발 키트(SDK)와 함께 사용하는 방법을 보여줍니다. 이 장의 코드 예제는 이 가이드의 나머지 부분에 있는 코드 예제를 보완하기 위한 것입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

코드 예시

- AWS SDKs를 사용한 Amazon Rekognition의 기본 예제
  - Amazon Rekognition 시작
  - AWS SDKs를 사용한 Amazon Rekognition 작업
    - AWS SDK 또는 CLI와 CompareFaces 함께 사용
    - AWS SDK 또는 CLI와 CreateCollection 함께 사용
    - AWS SDK 또는 CLI와 DeleteCollection 함께 사용
    - AWS SDK 또는 CLI와 DeleteFaces 함께 사용
    - AWS SDK 또는 CLI와 DescribeCollection 함께 사용
    - AWS SDK 또는 CLI와 DetectFaces 함께 사용
    - AWS SDK 또는 CLI와 DetectLabels 함께 사용
    - AWS SDK 또는 CLI와 DetectModerationLabels 함께 사용
    - AWS SDK 또는 CLI와 DetectText 함께 사용
    - AWS SDK 또는 CLI와 GetCelebrityInfo 함께 사용
    - AWS SDK 또는 CLI와 IndexFaces 함께 사용
    - AWS SDK 또는 CLI와 ListCollections 함께 사용
    - AWS SDK 또는 CLI와 ListFaces 함께 사용
    - AWS SDK 또는 CLI와 RecognizeCelebrities 함께 사용
    - AWS SDK 또는 CLI와 SearchFaces 함께 사용

    - AWS SDK 또는 CLI와 SearchFacesByImage 함께 사용
- AWS SDKs를 사용한 Amazon Rekognition 시나리오
  - AWS SDK를 사용하여 Amazon Rekognition 컬렉션을 빌드하고 해당 컬렉션에서 얼굴 찾기
  - 사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기
  - AWS SDK를 사용하여 Amazon Rekognition으로 이미지에서 PPE 감지
  - AWS SDK를 사용하여 Amazon Rekognition을 사용하여 이미지의 요소 감지 및 표시
  - AWS SDK를 사용하여 이미지에서 얼굴 감지
  - Amazon Rekognition 및 AWS SDK를 사용하여 비디오에서 정보 감지
  - AWS SDK를 사용하여 Amazon Rekognition으로 이미지의 객체 감지
  - AWS SDK를 사용하여 Amazon Rekognition을 사용하여 비디오에서 사람 및 객체 감지
  - AWS SDK를 사용하여 EXIF 및 기타 이미지 정보 저장

AWS SDKs를 사용한 Amazon Rekognition의 기본 예제

다음 코드 예제에서는 AWS SDK와 함께 Amazon Rekognition의 기본 사항을 사용하는 방법을 보여줍니다.

예제

- Amazon Rekognition 시작
- AWS SDKs를 사용한 Amazon Rekognition 작업
  - AWS SDK 또는 CLI와 CompareFaces 함께 사용
  - AWS SDK 또는 CLI와 CreateCollection 함께 사용
  - AWS SDK 또는 CLI와 DeleteCollection 함께 사용
  - AWS SDK 또는 CLI와 DeleteFaces 함께 사용
  - AWS SDK 또는 CLI와 DescribeCollection 함께 사용
  - AWS SDK 또는 CLI와 DetectFaces 함께 사용
  - AWS SDK 또는 CLI와 DetectLabels 함께 사용
  - AWS SDK 또는 CLI와 DetectModerationLabels 함께 사용
  - AWS SDK 또는 CLI와 DetectText 함께 사용
  - AWS SDK 또는 CLI와 GetCelebrityInfo 함께 사용
  - AWS SDK 또는 CLI와 IndexFaces 함께 사용

- AWS SDK 또는 CLI와 ListCollections 함께 사용
- AWS SDK 또는 CLI와 ListFaces 함께 사용
- AWS SDK 또는 CLI와 RecognizeCelebrities 함께 사용
- AWS SDK 또는 CLI와 SearchFaces 함께 사용
- AWS SDK 또는 CLI와 SearchFacesByImage 함께 사용

Amazon Rekognition 시작

다음 코드 예제에서는 Amazon Rekognition 사용을 시작하는 방법을 보여줍니다.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CMakeLists.txt CMake 파일의 코드입니다.

```
# 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 소스 파일의 코드입니다.

```
#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;
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 ListCollections를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDKs를 사용한 Amazon Rekognition 작업

다음 코드 예제에서는 AWS SDKs를 사용하여 개별 Amazon Rekognition 작업을 수행하는 방법을 보여줍니다. 각 예제에는 GitHub에 대한 링크가 포함되어 있습니다. 여기에서 코드 설정 및 실행에 대한 지침을 찾을 수 있습니다.

이들 발췌문은 Amazon Rekognition API를 직접적으로 호출하며, 컨텍스트에서 실행되어야 하는 더 큰 프로그램에서 발췌한 코드입니다. AWS SDKs를 사용한 Amazon Rekognition 시나리오에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 Amazon Rekognition API Reference를 참조하세요.

예제

- AWS SDK 또는 CLI와 CompareFaces 함께 사용
- AWS SDK 또는 CLI와 CreateCollection 함께 사용
- AWS SDK 또는 CLI와 DeleteCollection 함께 사용
- AWS SDK 또는 CLI와 DeleteFaces 함께 사용
- AWS SDK 또는 CLI와 DescribeCollection 함께 사용
- AWS SDK 또는 CLI와 DetectFaces 함께 사용
- AWS SDK 또는 CLI와 DetectLabels 함께 사용
- AWS SDK 또는 CLI와 DetectModerationLabels 함께 사용
- AWS SDK 또는 CLI와 DetectText 함께 사용
- AWS SDK 또는 CLI와 GetCelebrityInfo 함께 사용
- AWS SDK 또는 CLI와 IndexFaces 함께 사용
- AWS SDK 또는 CLI와 ListCollections 함께 사용
- AWS SDK 또는 CLI와 ListFaces 함께 사용
- AWS SDK 또는 CLI와 RecognizeCelebrities 함께 사용
- AWS SDK 또는 CLI와 SearchFaces 함께 사용
- AWS SDK 또는 CLI와 SearchFacesByImage 함께 사용

AWS SDK 또는 CLI와 **CompareFaces** 함께 사용

다음 코드 예시는 CompareFaces의 사용 방법을 보여 줍니다.

자세한 내용은 이미지에 있는 얼굴 비교를 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    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.");
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 CompareFaces를 참조하세요.

CLI

AWS CLI

두 이미지에서 얼굴을 비교하는 방법

다음 `compare-faces` 명령은 Amazon S3 버킷에 저장된 두 이미지에서 얼굴을 비교합니다.

```
aws rekognition compare-faces \
    --source-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"source.jpg"}}' \
    --target-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"target.jpg"}}'
```

출력:

```
{
    "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
    }
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 이미지 내 얼굴 비교를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CompareFaces 항목을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import software.amazon.awssdk.core.SdkBytes;

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

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

```
            Usage: <bucketName> <sourceKey> <targetKey>

            Where:
                bucketName - The name of the S3 bucket where the images are
stored.
                sourceKey  - The S3 key (file name) for the source image.
                targetKey  - The S3 key (file name) for the target image.
            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceKey = args[1];
        String targetKey = args[2];

        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();
        compareTwoFaces(rekClient, bucketName, sourceKey, targetKey);
    }

   /**
    * Compares two faces from images stored in an Amazon S3 bucket using AWS
Rekognition.
    *
    * <p>This method takes two image keys from an S3 bucket and compares the
faces within them.
    * It prints out the confidence level of matched faces and reports the number
of unmatched faces.</p>
    *
    * @param rekClient   The {@link RekognitionClient} used to call AWS
Rekognition.
    * @param bucketName  The name of the S3 bucket containing the images.
    * @param sourceKey   The object key (file path) for the source image in the
S3 bucket.
    * @param targetKey   The object key (file path) for the target image in the
S3 bucket.
    * @throws RuntimeException If the Rekognition service returns an error.
    */
```

```
   public static void compareTwoFaces(RekognitionClient rekClient, String
bucketName, String sourceKey, String targetKey) {
       try {
           Float similarityThreshold = 70F;
           S3Object s3ObjectSource = S3Object.builder()
                   .bucket(bucketName)
                   .name(sourceKey)
                   .build();

           Image sourceImage = Image.builder()
                   .s3Object(s3ObjectSource)
                   .build();

           S3Object s3ObjectTarget = S3Object.builder()
                   .bucket(bucketName)
                   .name(targetKey)
                   .build();

           Image targetImage = Image.builder()
                   .s3Object(s3ObjectTarget)
                   .build();

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

           // Compare the two images.
           CompareFacesResponse compareFacesResult =
rekClient.compareFaces(facesRequest);
           List<CompareFacesMatch> faceDetails =
compareFacesResult.faceMatches();

           for (CompareFacesMatch match : faceDetails) {
               ComparedFace face = match.face();
               BoundingBox position = face.boundingBox();
               System.out.println("Face at " + position.left().toString()
                       + " " + position.top()
                       + " matches with " + face.confidence().toString()
                       + "% confidence.");
           }
```

```
            List<ComparedFace> unmatchedFaces =
 compareFacesResult.unmatchedFaces();
            System.out.println("There were " + unmatchedFaces.size() + " face(s)
 that did not match.");

        } catch (RekognitionException e) {
            System.err.println("Error comparing faces: " +
 e.awsErrorDetails().errorMessage());
            throw new RuntimeException(e);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CompareFaces 항목을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun compareTwoFaces(
    similarityThresholdVal: Float,
    sourceImageVal: String,
    targetImageVal: String,
) {
    val sourceBytes = (File(sourceImageVal).readBytes())
    val targetBytes = (File(targetImageVal).readBytes())

    // Create an Image object for the source image.
    val souImage =
        Image {
            bytes = sourceBytes
        }

    val tarImage =
        Image {
```

```
            bytes = targetBytes
        }

    val facesRequest =
        CompareFacesRequest {
            sourceImage = souImage
            targetImage = tarImage
            similarityThreshold = similarityThresholdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->

        val compareFacesResult = rekClient.compareFaces(facesRequest)
        val faceDetails = compareFacesResult.faceMatches

        if (faceDetails != null) {
            for (match: CompareFacesMatch in faceDetails) {
                val face = match.face
                val position = face?.boundingBox
                if (position != null) {
                    println("Face at ${position.left} ${position.top} matches
 with ${face.confidence} % confidence.")
                }
            }
        }

        val uncompared = compareFacesResult.unmatchedFaces
        if (uncompared != null) {
            println("There was ${uncompared.size} face(s) that did not match")
        }

        println("Source image rotation:
 ${compareFacesResult.sourceImageOrientationCorrection}")
        println("target image rotation:
 ${compareFacesResult.targetImageOrientationCorrection}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CompareFaces를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def compare_faces(self, target_image, similarity):
        """
        Compares faces in the image with the largest face in the target image.

        :param target_image: The target image to compare against.
        :param similarity: Faces in the image must have a similarity value
 greater
                           than this value to be included in the results.
        :return: A tuple. The first element is the list of faces that match the
                 reference image. The second element is the list of faces that
 have
                 a similarity value below the specified threshold.
```

```
        """
        try:
            response = self.rekognition_client.compare_faces(
                SourceImage=self.image,
                TargetImage=target_image.image,
                SimilarityThreshold=similarity,
            )
            matches = [
                RekognitionFace(match["Face"]) for match in
response["FaceMatches"]
            ]
            unmatches = [RekognitionFace(face) for face in
response["UnmatchedFaces"]]
            logger.info(
                "Found %s matched faces and %s unmatched faces.",
                len(matches),
                len(unmatches),
            )
        except ClientError:
            logger.exception(
                "Couldn't match faces from %s to %s.",
                self.image_name,
                target_image.image_name,
            )
            raise
        else:
            return matches, unmatches
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CompareFaces를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Create S3 object reference for the source image
        DATA(lo_source_s3obj) = NEW /aws1/cl_reks3object(
          iv_bucket = iv_source_s3_bucket
          iv_name = iv_source_s3_key ).

        " Create source image object
        DATA(lo_source_image) = NEW /aws1/cl_rekimage(
          io_s3object = lo_source_s3obj ).

        " Create S3 object reference for the target image
        DATA(lo_target_s3obj) = NEW /aws1/cl_reks3object(
          iv_bucket = iv_target_s3_bucket
          iv_name = iv_target_s3_key ).

        " Create target image object
        DATA(lo_target_image) = NEW /aws1/cl_rekimage(
          io_s3object = lo_target_s3obj ).

        " Compare faces
        oo_result = lo_rek->comparefaces(
          io_sourceimage = lo_source_image
          io_targetimage = lo_target_image
          iv_similaritythreshold = iv_similarity ).

        DATA(lt_face_matches) = oo_result->get_facematches( ).
        DATA(lt_unmatched_faces) = oo_result->get_unmatchedfaces( ).

        " Get counts of matched and unmatched faces
        DATA(lv_matched_count) = lines( lt_face_matches ).
        DATA(lv_unmatched_count) = lines( lt_unmatched_faces ).

        " Output detailed comparison results
        DATA(lv_message) = |Face comparison completed: | &&
                           |{ lv_matched_count } matched face(s), | &&
                           |{ lv_unmatched_count } unmatched face(s).|.
        MESSAGE lv_message TYPE 'I'.
      CATCH /aws1/cx_rekinvalids3objectex.
        MESSAGE 'Invalid S3 object.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 CompareFaces를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `CreateCollection` 함께 사용

다음 코드 예시는 `CreateCollection`의 사용 방법을 보여 줍니다.

자세한 내용은 컬렉션 생성을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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}");
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 CreateCollection을 참조하세요.

CLI

AWS CLI

모음을 생성하는 방법

다음 `create-collection` 명령은 지정된 이름을 가진 컬렉션을 생성합니다.

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

출력:

```
{
    "CollectionArn": "aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0",
    "StatusCode": 200
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 컬렉션 생성을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CreateCollection을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.CreateCollectionRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

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

            Usage: <collectionName>\s

            Where:
                collectionName - The name of the collection.\s
            """;

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

        String collectionId = args[0];
```

```
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Creating collection: " + collectionId);
        createMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    /**
     * Creates a new Amazon Rekognition collection.
     *
     * @param rekClient    the Amazon Rekognition client used to interact with
 the Rekognition service
     * @param collectionId the unique identifier for the collection to be created
     */
    public static void createMyCollection(RekognitionClient rekClient, String
 collectionId) {
        try {
            CreateCollectionRequest collectionRequest =
 CreateCollectionRequest.builder()
                    .collectionId(collectionId)
                    .build();

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateCollection을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createMyCollection(collectionIdVal: String) {
    val request =
        CreateCollectionRequest {
            collectionId = collectionIdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.createCollection(request)
        println("Collection ARN is ${response.collectionArn}")
        println("Status code is ${response.statusCode}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateCollection을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
```

```
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client


    def create_collection(self, collection_id):
        """
        Creates an empty collection.

        :param collection_id: Text that identifies the collection.
        :return: The newly created collection.
        """
        try:
            response = self.rekognition_client.create_collection(
                CollectionId=collection_id
            )
            response["CollectionId"] = collection_id
            collection = RekognitionCollection(response, self.rekognition_client)
            logger.info("Created collection %s.", collection_id)
        except ClientError:
            logger.exception("Couldn't create collection %s.", collection_id)
            raise
        else:
            return collection
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 CreateCollection을 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
TRY.
    oo_result = lo_rek->createcollection(
      iv_collectionid = iv_collection_id ).
    MESSAGE 'Collection created successfully.' TYPE 'I'.
  CATCH /aws1/cx_rekresrcalrdyexistsex.
    MESSAGE 'Collection already exists.' TYPE 'E'.
  CATCH /aws1/cx_rekinvalidparameterex.
    MESSAGE 'Invalid parameter value.' TYPE 'E'.
ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 CreateCollection을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `DeleteCollection` 함께 사용

다음 코드 예시는 `DeleteCollection`의 사용 방법을 보여 줍니다.

자세한 내용은 컬렉션 삭제를 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    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}");
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DeleteCollection을 참조하세요.

CLI

AWS CLI

모음을 삭제하는 방법

다음 `delete-collection` 명령은 지정된 컬렉션을 삭제합니다.

```
aws rekognition delete-collection \
```

```
    --collection-id MyCollection
```

출력:

```
{
    "StatusCode": 200
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 컬렉션 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteCollection을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

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

```
            Where:
                collectionId - The id of the collection to delete.\s
            """;

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

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

        System.out.println("Deleting collection: " + collectionId);
        deleteMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    /**
     * Deletes an Amazon Rekognition collection.
     *
     * @param rekClient      An instance of the {@link RekognitionClient} class,
which is used to interact with the Amazon Rekognition service.
     * @param collectionId   The ID of the collection to be deleted.
     */
    public static void deleteMyCollection(RekognitionClient rekClient, String
collectionId) {
        try {
            DeleteCollectionRequest deleteCollectionRequest =
DeleteCollectionRequest.builder()
                    .collectionId(collectionId)
                    .build();

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

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

```
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteCollection을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteMyCollection(collectionIdVal: String) {
    val request =
        DeleteCollectionRequest {
            collectionId = collectionIdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.deleteCollection(request)
        println("The collectionId status is ${response.statusCode}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteCollection을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
```

```
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def delete_collection(self):
        """
        Deletes the collection.
        """
        try:

self.rekognition_client.delete_collection(CollectionId=self.collection_id)
            logger.info("Deleted collection %s.", self.collection_id)
            self.collection_id = None
        except ClientError:
            logger.exception("Couldn't delete collection %s.",
self.collection_id)
            raise
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 DeleteCollection을 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        lo_rek->deletecollection(
          iv_collectionid = iv_collection_id ).
        MESSAGE 'Collection deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_rekresourcenotfoundex.
        MESSAGE 'Collection not found.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 DeleteCollection을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `DeleteFaces` 함께 사용

다음 코드 예시는 DeleteFaces의 사용 방법을 보여 줍니다.

자세한 내용은 컬렉션에서 얼굴 삭제를 참조하십시오.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    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}");
            });
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DeleteFaces를 참조하세요.

CLI

AWS CLI

모음에서 얼굴을 삭제하는 방법

다음 `delete-faces` 명령은 컬렉션에서 지정된 얼굴을 삭제합니다.

```
aws rekognition delete-faces \
    --collection-id MyCollection
    --face-ids '["0040279c-0178-436e-b70a-e61b074e96b0"]'
```

출력:

```
{
    "DeletedFaces": [
        "0040279c-0178-436e-b70a-e61b074e96b0"
    ]
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 컬렉션에서 얼굴 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteFaces를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteFacesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

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

            Where:
                collectionId - The id of the collection from which faces are
 deleted.\s
                faceId - The id of the face to delete.\s
           """;

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

        String collectionId = args[0];
        String faceId = args[1];
```

```
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteFacesCollection(rekClient, collectionId, faceId);
        rekClient.close();
    }

    /**
     * Deletes a face from the specified Amazon Rekognition collection.
     *
     * @param rekClient     an instance of the Amazon Rekognition client
     * @param collectionId  the ID of the collection from which the face should
 be deleted
     * @param faceId        the ID of the face to be deleted
     * @throws RekognitionException if an error occurs while deleting the face
     */
    public static void deleteFacesCollection(RekognitionClient rekClient,
            String collectionId,
            String faceId) {

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

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

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteFaces를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteFacesCollection(
    collectionIdVal: String?,
    faceIdVal: String,
) {
    val deleteFacesRequest =
        DeleteFacesRequest {
            collectionId = collectionIdVal
            faceIds = listOf(faceIdVal)
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        rekClient.deleteFaces(deleteFacesRequest)
        println("$faceIdVal was deleted from the collection")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteFaces를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollection:
```

```
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def delete_faces(self, face_ids):
        """
        Deletes faces from the collection.

        :param face_ids: The list of IDs of faces to delete.
        :return: The list of IDs of faces that were deleted.
        """
        try:
```

```
            response = self.rekognition_client.delete_faces(
                CollectionId=self.collection_id, FaceIds=face_ids
            )
            deleted_ids = response["DeletedFaces"]
            logger.info(
                "Deleted %s faces from %s.", len(deleted_ids), self.collection_id
            )
        except ClientError:
            logger.exception("Couldn't delete faces from %s.",
self.collection_id)
            raise
        else:
            return deleted_ids
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 DeleteFaces를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_rek->deletefaces(
          iv_collectionid = iv_collection_id
          it_faceids = it_face_ids ).

        DATA(lt_deleted_faces) = oo_result->get_deletedfaces( ).
        DATA(lv_deleted_count) = lines( lt_deleted_faces ).
        DATA(lv_msg6) = |{ lv_deleted_count } face(s) deleted successfully.|.
        MESSAGE lv_msg6 TYPE 'I'.
      CATCH /aws1/cx_rekresourcenotfoundex.
        MESSAGE 'Collection not found.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
```

```
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 DeleteFaces를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DescribeCollection** 함께 사용

다음 코드 예시는 DescribeCollection의 사용 방법을 보여 줍니다.

자세한 내용은 컬렉션 설명을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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}");
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DescribeCollection을 참조하세요.

CLI

AWS CLI

모음을 설명하는 방법

다음 `describe-collection` 예시에서는 지정된 컬렉션의 세부 정보를 표시합니다.

```
aws rekognition describe-collection \
    --collection-id MyCollection
```

출력:

```
{
    "FaceCount": 200,
    "CreationTimestamp": 1569444828.274,
    "CollectionARN": "arn:aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0"
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 컬렉션 설명을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DescribeCollection을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

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

            Where:
                collectionName - The name of the Amazon Rekognition collection.\s
            """;

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

```
            System.exit(1);
        }

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

        describeColl(rekClient, collectionName);
        rekClient.close();
    }

    /**
     * Describes an Amazon Rekognition collection.
     *
     * @param rekClient         The Amazon Rekognition client used to make the
 request.
     * @param collectionName    The name of the collection to describe.
     *
     * @throws RekognitionException If an error occurs while describing the
 collection.
     */
    public static void describeColl(RekognitionClient rekClient, String
 collectionName) {
        try {
            DescribeCollectionRequest describeCollectionRequest =
 DescribeCollectionRequest.builder()
                    .collectionId(collectionName)
                    .build();

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeCollection을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun describeColl(collectionName: String) {
    val request =
        DescribeCollectionRequest {
            collectionId = collectionName
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.describeCollection(request)
        println("The collection Arn is ${response.collectionArn}")
        println("The collection contains this many faces ${response.faceCount}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeCollection을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def describe_collection(self):
        """
        Gets data about the collection from the Amazon Rekognition service.

        :return: The collection rendered as a dict.
        """
```

```
        try:
            response = self.rekognition_client.describe_collection(
                CollectionId=self.collection_id
            )
            # Work around capitalization of Arn vs. ARN
            response["CollectionArn"] = response.get("CollectionARN")
            (
                self.collection_arn,
                self.face_count,
                self.created,
            ) = self._unpack_collection(response)
            logger.info("Got data for collection %s.", self.collection_id)
        except ClientError:
            logger.exception("Couldn't get data for collection %s.",
self.collection_id)
            raise
        else:
            return self.to_dict()
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 DescribeCollection을 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_rek->describecollection(
          iv_collectionid = iv_collection_id ).
        DATA(lv_face_count) = oo_result->get_facecount( ).
        DATA(lv_msg) = |Collection described: { lv_face_count } face(s)
indexed.|.
        MESSAGE lv_msg TYPE 'I'.
      CATCH /aws1/cx_rekresourcenotfoundex.
        MESSAGE 'Collection not found.' TYPE 'E'.
```

```
    CATCH /aws1/cx_rekinvalidparameterex.
      MESSAGE 'Invalid parameter value.' TYPE 'E'.
ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 DescribeCollection을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `DetectFaces` 함께 사용

다음 코드 예시는 DetectFaces의 사용 방법을 보여 줍니다.

자세한 내용은 이미지에서 얼굴 감지를 참조하십시오.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectFaces
{
    public static async Task Main()
    {
        string photo = "input.jpg";
        string bucket = "amzn-s3-demo-bucket";
```

```
            var rekognitionClient = new AmazonRekognitionClient();

            var detectFacesRequest = new DetectFacesRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },

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

            try
            {
                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
                foreach (FaceDetail face in detectFacesResponse.FaceDetails)
                {
                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"Confidence: {face.Confidence}");
                    Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
                    Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
 roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
                    Console.WriteLine($"Brightness:
 {face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

                    if (hasAll)
                    {
                        Console.WriteLine($"Estimated age is between
 {face.AgeRange.Low} and {face.AgeRange.High} years old.");
                    }
```

```
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

이미지 내 모든 얼굴에 대한 경계 상자 정보를 표시합니다.

```
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to display the details of the
    /// bounding boxes around the faces detected in an image.
    /// </summary>
    public class ImageOrientationBoundingBox
    {
        public static async Task Main()
        {
            string photo = @"D:\Development\AWS-Examples\Rekognition
\target.jpg"; // "photo.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
```

```
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            int height;
            int width;

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

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

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

                DetectFacesResponse detectFacesResponse = await
rekognitionClient.DetectFacesAsync(detectFacesRequest);
                detectFacesResponse.FaceDetails.ForEach(face =>
                {
                    Console.WriteLine("Face:");
                    ShowBoundingBoxPositions(
                        height,
                        width,
                        face.BoundingBox,
                        detectFacesResponse.OrientationCorrection);

                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
left={face.BoundingBox.Top} width={face.BoundingBox.Width}
height={face.BoundingBox.Height}");
```

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

        /// <summary>
        /// Display the bounding box information for an image.
        /// </summary>
        /// <param name="imageHeight">The height of the image.</param>
        /// <param name="imageWidth">The width of the image.</param>
        /// <param name="box">The bounding box for a face found within the
 image.</param>
        /// <param name="rotation">The rotation of the face's bounding box.</
param>
        public static void ShowBoundingBoxPositions(int imageHeight, int
 imageWidth, BoundingBox box, string rotation)
        {
            float left;
            float top;

            if (rotation == null)
            {
                Console.WriteLine("No estimated orientation. Check Exif data.");
                return;
            }

            // Calculate face position based on image orientation.
            switch (rotation)
            {
                case "ROTATE_0":
                    left = imageWidth * box.Left;
                    top = imageHeight * box.Top;
                    break;
                case "ROTATE_90":
                    left = imageHeight * (1 - (box.Top + box.Height));
                    top = imageWidth * box.Left;
                    break;
                case "ROTATE_180":
                    left = imageWidth - (imageWidth * (box.Left + box.Width));
```

```
                    top = imageHeight * (1 - (box.Top + box.Height));
                    break;
                case "ROTATE_270":
                    left = imageHeight * box.Top;
                    top = imageWidth * (1 - box.Left - box.Width);
                    break;
                default:
                    Console.WriteLine("No estimated orientation information.
 Check Exif data.");
                    return;
            }

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DetectFaces를 참조하세요.

CLI

AWS CLI

이미지에서 얼굴을 감지하는 방법

다음 `detect-faces` 명령은 Amazon S3 버킷에 저장된 지정된 이미지에서 얼굴을 감지합니다.

```
aws rekognition detect-faces \
    --image '{"S3Object":{"Bucket":"MyImageS3Bucket","Name":"MyFriend.jpg"}}' \
    --attributes "ALL"
```

출력:

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

자세한 내용은 Amazon Rekognition 개발자 안내서의 이미지에서 얼굴 감지를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DetectFaces를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.util.List;
```

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

            Usage:   <bucketName> <sourceImage>

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

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

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

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

    /**
     * Detects faces in an image stored in an Amazon S3 bucket using the Amazon
 Rekognition service.
     *
```

```
    * @param rekClient    The Amazon Rekognition client used to interact with
the Rekognition service.
    * @param bucketName   The name of the Amazon S3 bucket where the source
image is stored.
    * @param sourceImage  The name of the source image file in the Amazon S3
bucket.
    */
   public static void detectFacesinImage(RekognitionClient rekClient, String
bucketName, String sourceImage) {
       try {
           S3Object s3ObjectTarget = S3Object.builder()
               .bucket(bucketName)
               .name(sourceImage)
               .build();

           Image targetImage = Image.builder()
               .s3Object(s3ObjectTarget)
               .build();

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

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

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

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

```
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DetectFaces를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun detectFacesinImage(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectFacesRequest {
            attributes = listOf(Attribute.All)
            image = souImage
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectFaces(request)
        response.faceDetails?.forEach { face ->
            val ageRange = face.ageRange
            println("The detected face is estimated to be between
 ${ageRange?.low} and ${ageRange?.high} years old.")
            println("There is a smile ${face.smile?.value}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DetectFaces를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_faces(self):
        """
        Detects faces in the image.

        :return: The list of faces found in the image.
        """
        try:
            response = self.rekognition_client.detect_faces(
                Image=self.image, Attributes=["ALL"]
            )
            faces = [RekognitionFace(face) for face in response["FaceDetails"]]
            logger.info("Detected %s faces.", len(faces))
```

```
        except ClientError:
            logger.exception("Couldn't detect faces in %s.", self.image_name)
            raise
        else:
            return faces
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DetectFaces를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
TRY.
    " Create S3 object reference for the image
    DATA(lo_s3object) = NEW /aws1/cl_reks3object(
      iv_bucket = iv_s3_bucket
      iv_name = iv_s3_key ).

    " Create image object
    DATA(lo_image) = NEW /aws1/cl_rekimage(
      io_s3object = lo_s3object ).

    " Detect faces in the image with all attributes
    DATA(lt_attributes) = VALUE /aws1/cl_rekattributes_w=>tt_attributes( ).
    DATA(lo_attr_wrapper) = NEW /aws1/cl_rekattributes_w( iv_value = 'ALL' ).
    INSERT lo_attr_wrapper INTO TABLE lt_attributes.

    oo_result = lo_rek->detectfaces(
      io_image = lo_image
      it_attributes = lt_attributes ).

    DATA(lt_face_details) = oo_result->get_facedetails( ).
    DATA(lv_detected_count) = lines( lt_face_details ).
    DATA(lv_msg8) = |{ lv_detected_count } face(s) detected in image.|.
```

```
      MESSAGE lv_msg8 TYPE 'I'.
    CATCH /aws1/cx_rekinvalids3objectex.
      MESSAGE 'Invalid S3 object.' TYPE 'E'.
    CATCH /aws1/cx_rekinvalidparameterex.
      MESSAGE 'Invalid parameter value.' TYPE 'E'.
  ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 DetectFaces를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **`DetectLabels`** 함께 사용

다음 코드 예시는 `DetectLabels`의 사용 방법을 보여 줍니다.

자세한 내용은 이미지에서 레이블 감지를 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect labels within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectLabels
{
    public static async Task Main()
```

```
        {
            string photo = "del_river_02092020_01.jpg"; // "input.jpg";
            string bucket = "amzn-s3-demo-bucket"; // "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

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

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

컴퓨터에 저장된 이미지 파일에서 레이블을 감지합니다.

```
    using System;
```

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored locally.
    /// </summary>
    public class DetectLabelsLocalFile
    {
        public static async Task Main()
        {
            string photo = "input.jpg";

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

            var rekognitionClient = new AmazonRekognitionClient();

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

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
```

```
                Console.WriteLine($"Detected labels for {photo}");
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"{label.Name}: {label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DetectLabels 항목을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! 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();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 DetectLabels를 참조하세요.

CLI

AWS CLI

이미지에서 레이블을 감지하는 방법

다음 detect-labels 예시에서는 Amazon S3 버킷에 저장된 이미지에서 장면과 객체를 감지합니다.

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

출력:

```
{
    "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"
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 이미지에서 레이블 감지를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DetectLabels를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

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

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

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

        if (args.length != 2) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

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

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

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

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

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

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DetectLabels 항목을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun detectImageLabels(sourceImage: String) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }
    val request =
        DetectLabelsRequest {
            image = souImage
            maxLabels = 10
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectLabels(request)
```

```
        response.labels?.forEach { label ->
            println("${label.name} : ${label.confidence}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DetectLabels를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.
```

```
        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels
            )
            labels = [RekognitionLabel(label) for label in response["Labels"]]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 DetectLabels를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Create S3 object reference for the image
        DATA(lo_s3object) = NEW /aws1/cl_reks3object(
          iv_bucket = iv_s3_bucket
          iv_name = iv_s3_key ).

        " Create image object
        DATA(lo_image) = NEW /aws1/cl_rekimage(
          io_s3object = lo_s3object ).

        " Detect labels in the image
        oo_result = lo_rek->detectlabels(
```

```
          io_image = lo_image
          iv_maxlabels = iv_max_labels ).

        DATA(lt_labels) = oo_result->get_labels( ).
        DATA(lv_label_count) = lines( lt_labels ).
        DATA(lv_msg9) = |{ lv_label_count } label(s) detected in image.|.
        MESSAGE lv_msg9 TYPE 'I'.
      CATCH /aws1/cx_rekinvalids3objectex.
        MESSAGE 'Invalid S3 object.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 DetectLabels를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `DetectModerationLabels` 함께 사용

다음 코드 예시는 `DetectModerationLabels`의 사용 방법을 보여 줍니다.

자세한 내용은 부적절한 이미지 감지를 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect unsafe content in a
```

```
   /// JPEG or PNG format image.
   /// </summary>
   public class DetectModerationLabels
   {
       public static async Task Main(string[] args)
       {
           string photo = "input.jpg";
           string bucket = "amzn-s3-demo-bucket";

           var rekognitionClient = new AmazonRekognitionClient();

           var detectModerationLabelsRequest = new
DetectModerationLabelsRequest()
           {
               Image = new Image()
               {
                   S3Object = new S3Object()
                   {
                       Name = photo,
                       Bucket = bucket,
                   },
               },
               MinConfidence = 60F,
           };

           try
           {
               var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
               Console.WriteLine("Detected labels for " + photo);
               foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
               {
                   Console.WriteLine($"Label: {label.Name}");
                   Console.WriteLine($"Confidence: {label.Confidence}");
                   Console.WriteLine($"Parent: {label.ParentName}");
               }
           }
           catch (Exception ex)
           {
               Console.WriteLine(ex.Message);
           }
       }
   }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DetectModerationLabels를 참조하세요.

CLI

AWS CLI

이미지에서 안전하지 않은 콘텐츠를 감지하는 방법

다음 `detect-moderation-labels` 명령은 Amazon S3 버킷에 저장된 지정된 이미지에서 안전하지 않은 콘텐츠를 감지합니다.

```
aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=gun.jpg}"
```

출력:

```
{
    "ModerationModelVersion": "3.0",
    "ModerationLabels": [
        {
            "Confidence": 97.29618072509766,
            "ParentName": "Violence",
            "Name": "Weapon Violence"
        },
        {
            "Confidence": 97.29618072509766,
            "ParentName": "",
            "Name": "Violence"
        }
    ]
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 안전하지 않은 이미지 감지를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DetectModerationLabels를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

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

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

    public static void main(String[] args) {
        final String usage = """
            Usage:  <bucketName>  <sourceImage>

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

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

```
            System.exit(1);
        }

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

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

    /**
     * Detects moderation labels in an image stored in an Amazon S3 bucket.
     *
     * @param rekClient      the Amazon Rekognition client to use for the
detection
     * @param bucketName     the name of the Amazon S3 bucket where the image is
stored
     * @param sourceImage    the name of the image file to be analyzed
     *
     * @throws RekognitionException if there is an error during the image
detection process
     */
    public static void detectModLabels(RekognitionClient rekClient, String
bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image targetImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            DetectModerationLabelsRequest moderationLabelsRequest =
DetectModerationLabelsRequest.builder()
                    .image(targetImage)
                    .minConfidence(60F)
                    .build();
```

```
            DetectModerationLabelsResponse moderationLabelsResponse = rekClient
                    .detectModerationLabels(moderationLabelsRequest);
            List<ModerationLabel> labels =
 moderationLabelsResponse.moderationLabels();
            System.out.println("Detected labels for image");
            for (ModerationLabel label : labels) {
                System.out.println("Label: " + label.name()
                        + "\n Confidence: " + label.confidence().toString() + "%"
                        + "\n Parent:" + label.parentName());
            }

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DetectModerationLabels를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun detectModLabels(sourceImage: String) {
    val myImage =
        Image {
            this.bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectModerationLabelsRequest {
            image = myImage
            minConfidence = 60f
        }
```

```
    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectModerationLabels(request)
        response.moderationLabels?.forEach { label ->
            println("Label: ${label.name} - Confidence: ${label.confidence} %
 Parent: ${label.parentName}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DetectModerationLabels를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client
```

```
    def detect_moderation_labels(self):
        """
        Detects moderation labels in the image. Moderation labels identify
content
        that may be inappropriate for some audiences.

        :return: The list of moderation labels found in the image.
        """
        try:
            response = self.rekognition_client.detect_moderation_labels(
                Image=self.image
            )
            labels = [
                RekognitionModerationLabel(label)
                for label in response["ModerationLabels"]
            ]
            logger.info(
                "Found %s moderation labels in %s.", len(labels), self.image_name
            )
        except ClientError:
            logger.exception(
                "Couldn't detect moderation labels in %s.", self.image_name
            )
            raise
        else:
            return labels
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 DetectModerationLabels를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Create S3 object reference for the image
        DATA(lo_s3object) = NEW /aws1/cl_reks3object(
          iv_bucket = iv_s3_bucket
          iv_name = iv_s3_key ).

        " Create image object
        DATA(lo_image) = NEW /aws1/cl_rekimage(
          io_s3object = lo_s3object ).

        " Detect moderation labels
        oo_result = lo_rek->detectmoderationlabels(
          io_image = lo_image ).

        DATA(lt_moderation_labels) = oo_result->get_moderationlabels( ).
        DATA(lv_mod_count) = lines( lt_moderation_labels ).
        DATA(lv_msg10) = |{ lv_mod_count } moderation label(s) detected.|.
        MESSAGE lv_msg10 TYPE 'I'.
      CATCH /aws1/cx_rekinvalids3objectex.
        MESSAGE 'Invalid S3 object.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 SDK for SAP ABAP API 참조의 DetectModerationLabels를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **`DetectText`** 함께 사용

다음 코드 예시는 `DetectText`의 사용 방법을 보여 줍니다.

자세한 내용은 이미지에서 텍스트 감지를 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect text in an image. The
/// example was created using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class DetectText
{
    public static async Task Main()
    {
        string photo = "Dad_photographer.jpg"; // "input.jpg";
        string bucket = "amzn-s3-demo-bucket"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
        };

        try
```

```
            {
                DetectTextResponse detectTextResponse = await
 rekognitionClient.DetectTextAsync(detectTextRequest);
                Console.WriteLine($"Detected lines and words for {photo}");
                detectTextResponse.TextDetections.ForEach(text =>
                {
                    Console.WriteLine($"Detected: {text.DetectedText}");
                    Console.WriteLine($"Confidence: {text.Confidence}");
                    Console.WriteLine($"Id : {text.Id}");
                    Console.WriteLine($"Parent Id: {text.ParentId}");
                    Console.WriteLine($"Type: {text.Type}");
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DetectText를 참조하세요.

CLI

AWS CLI

이미지에서 텍스트를 감지하는 방법

다음 `detect-text` 명령은 지정된 이미지에서 텍스트를 감지합니다.

```
aws rekognition detect-text \
    --image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"ExamplePicture.jpg"}}'
```

출력:

```
{
    "TextDetections": [
        {
            "Geometry": {
                "BoundingBox": {
```

```
                    "Width": 0.24624845385551453,
                    "Top": 0.28288066387176514,
                    "Left": 0.391388863325119,
                    "Height": 0.022687450051307678
                },
                "Polygon": [
                    {
                        "Y": 0.28288066387176514,
                        "X": 0.391388863325119
                    },
                    {
                        "Y": 0.2826388478279114,
                        "X": 0.6376373171806335
                    },
                    {
                        "Y": 0.30532628297805786,
                        "X": 0.637677013874054
                    },
                    {
                        "Y": 0.305568128824234,
                        "X": 0.39142853021621704
                    }
                ]
            },
            "Confidence": 94.35709381103516,
            "DetectedText": "ESTD 1882",
            "Type": "LINE",
            "Id": 0
        },
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933889865875244,
                    "Top": 0.32603850960731506,
                    "Left": 0.34534579515457153,
                    "Height": 0.07126858830451965
                },
                "Polygon": [
                    {
                        "Y": 0.32603850960731506,
                        "X": 0.34534579515457153
                    },
                    {
                        "Y": 0.32633158564567566,
```

```
                    "X": 0.684684693813324
                },
                {
                    "Y": 0.3976001739501953,
                    "X": 0.684575080871582
                },
                {
                    "Y": 0.3973070979118347,
                    "X": 0.345236212015152
                }
            ]
        },
        "Confidence": 99.95779418945312,
        "DetectedText": "BRAINS",
        "Type": "LINE",
        "Id": 1
    },
    {
        "Confidence": 97.22098541259766,
        "Geometry": {
            "BoundingBox": {
                "Width": 0.061079490929841995,
                "Top": 0.2843210697174072,
                "Left": 0.391391396522522,
                "Height": 0.021029088646173477
            },
            "Polygon": [
                {
                    "Y": 0.2843210697174072,
                    "X": 0.391391396522522
                },
                {
                    "Y": 0.2828207015991211,
                    "X": 0.4524524509906769
                },
                {
                    "Y": 0.3038259446620941,
                    "X": 0.4534534513950348
                },
                {
                    "Y": 0.30532634258270264,
                    "X": 0.3923923969268799
                }
            ]
```

```
            },
            "DetectedText": "ESTD",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 2
        },
        {
            "Confidence": 91.49320983886719,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.07007007300853729,
                    "Top": 0.2828207015991211,
                    "Left": 0.5675675868988037,
                    "Height": 0.02250562608242035
                },
                "Polygon": [
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.5675675868988037
                    },
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.5675675868988037
                    }
                ]
            },
            "DetectedText": "1882",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 3
        },
        {
            "Confidence": 99.95779418945312,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933934569358826,
```

```
                    "Top": 0.32633158564567566,
                    "Left": 0.3453453481197357,
                    "Height": 0.07127484679222107
                },
                "Polygon": [
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.3453453481197357
                    },
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.684684693813324
                    },
                    {
                        "Y": 0.39759939908981323,
                        "X": 0.6836836934089661
                    },
                    {
                        "Y": 0.39684921503067017,
                        "X": 0.3453453481197357
                    }
                ]
            },
            "DetectedText": "BRAINS",
            "ParentId": 1,
            "Type": "WORD",
            "Id": 4
        }
    ]
}
```


- API 세부 정보는 AWS CLI 명령 참조의 DetectText를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

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

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectText {
    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage:   <bucketName> <sourceImage>\n" +
            "\n" +
            "Where:\n" +
            "   bucketName - The name of the S3 bucket where the image is stored
\n" +
            "   sourceImage - The path to the image that contains text (for
 example, pic1.png). \n";

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

        String bucketName = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectTextLabels(rekClient, bucketName, sourceImage);
```

```
        rekClient.close();
    }

    /**
     * Detects text labels in an image stored in an S3 bucket using Amazon
Rekognition.
     *
     * @param rekClient    an instance of the Amazon Rekognition client
     * @param bucketName   the name of the S3 bucket where the image is stored
     * @param sourceImage  the name of the image file in the S3 bucket
     * @throws RekognitionException if an error occurs while calling the Amazon
Rekognition API
     */
    public static void detectTextLabels(RekognitionClient rekClient, String
bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            DetectTextRequest textRequest = DetectTextRequest.builder()
                    .image(souImage)
                    .build();

            DetectTextResponse textResponse = rekClient.detectText(textRequest);
            List<TextDetection> textCollection = textResponse.textDetections();
            System.out.println("Detected lines and words");
            for (TextDetection text : textCollection) {
                System.out.println("Detected: " + text.detectedText());
                System.out.println("Confidence: " +
text.confidence().toString());
                System.out.println("Id : " + text.id());
                System.out.println("Parent Id: " + text.parentId());
                System.out.println("Type: " + text.type());
                System.out.println();
            }

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

```
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DetectText를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun detectTextLabels(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectTextRequest {
            image = souImage
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectText(request)
        response.textDetections?.forEach { text ->
            println("Detected: ${text.detectedText}")
            println("Confidence: ${text.confidence}")
            println("Id: ${text.id}")
            println("Parent Id:  ${text.parentId}")
            println("Type: ${text.type}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DetectText를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [RekognitionText(text) for text in
 response["TextDetections"]]
```

```
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 DetectText를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
TRY.
    " Create S3 object reference for the image
    DATA(lo_s3object) = NEW /aws1/cl_reks3object(
      iv_bucket = iv_s3_bucket
      iv_name = iv_s3_key ).

    " Create image object
    DATA(lo_image) = NEW /aws1/cl_rekimage(
      io_s3object = lo_s3object ).

    " Detect text in the image
    oo_result = lo_rek->detecttext(
      io_image = lo_image ).

    DATA(lt_text_detections) = oo_result->get_textdetections( ).
    DATA(lv_text_count) = lines( lt_text_detections ).
    DATA(lv_msg11) = |{ lv_text_count } text detection(s) found.|.
    MESSAGE lv_msg11 TYPE 'I'.
  CATCH /aws1/cx_rekinvalids3objectex.
    MESSAGE 'Invalid S3 object.' TYPE 'E'.
  CATCH /aws1/cx_rekinvalidparameterex.
```

```
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 DetectText를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **GetCelebrityInfo** 함께 사용

다음 코드 예시는 GetCelebrityInfo의 사용 방법을 보여 줍니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Shows how to use Amazon Rekognition to retrieve information about the
/// celebrity identified by the supplied celebrity Id.
/// </summary>
public class CelebrityInfo
{
    public static async Task Main()
    {
        string celebId = "nnnnnnnn";

        var rekognitionClient = new AmazonRekognitionClient();

        var celebrityInfoRequest = new GetCelebrityInfoRequest
```

```
            {
                Id = celebId,
            };

            Console.WriteLine($"Getting information for celebrity: {celebId}");

            var celebrityInfoResponse = await
 rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

            // Display celebrity information.
            Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
            Console.WriteLine("Further information (if available):");
            celebrityInfoResponse.Urls.ForEach(url =>
            {
                Console.WriteLine(url);
            });
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetCelebrityInfo를 참조하세요.

CLI

AWS CLI

유명 인사에 대한 정보를 가져오는 방법

다음 `get-celebrity-info` 명령은 지정된 유명인의 정보를 표시합니다. `id` 파라미터는 이전 `recognize-celebrities` 직접 호출에서 가져온 것입니다.

```
aws rekognition get-celebrity-info --id nnnnnnn
```

출력:

```
{
    "Name": "Celeb A",
    "Urls": [
        "www.imdb.com/name/aaaaaaaaa"
    ]
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 유명인의 정보 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetCelebrityInfo를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `IndexFaces` 함께 사용

다음 코드 예시는 IndexFaces의 사용 방법을 보여 줍니다.

자세한 내용은 컬렉션에 얼굴 추가를 참조하십시오.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces in an image
/// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
/// bucket and then adds the information to a collection.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "amzn-s3-demo-bucket";
        string photo = "input.jpg";
```

```
            var rekognitionClient = new AmazonRekognitionClient();

            var image = new Image
            {
                S3Object = new S3Object
                {
                    Bucket = bucket,
                    Name = photo,
                },
            };

            var indexFacesRequest = new IndexFacesRequest
            {
                Image = image,
                CollectionId = collectionId,
                ExternalImageId = photo,
                DetectionAttributes = new List<string>() { "ALL" },
            };

            IndexFacesResponse indexFacesResponse = await
rekognitionClient.IndexFacesAsync(indexFacesRequest);

            Console.WriteLine($"{photo} added");
            foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
            {
                Console.WriteLine($"Face detected: Faceid is
{faceRecord.Face.FaceId}");
            }
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 IndexFaces를 참조하세요.

CLI

AWS CLI

모음에 얼굴을 추가하는 방법

다음 `index-faces` 명령은 이미지에서 찾은 얼굴을 지정된 컬렉션에 추가합니다.

```
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"
```

출력:

```
{
    "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"
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 컬렉션에 얼굴 추가를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 IndexFaces를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;

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

            Where:
```

```
                collectionName - The name of the collection.
                sourceImage - The name of the image (for example, pic1.png).
                bucketName - The name of the S3 bucket.
            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
        String bucketName = args[2];;
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        addToCollection(rekClient, collectionId, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Adds a face from an image to an Amazon Rekognition collection.
     *
     * @param rekClient     the Amazon Rekognition client
     * @param collectionId  the ID of the collection to add the face to
     * @param bucketName    the name of the Amazon S3 bucket containing the image
     * @param sourceImage   the name of the image file to add to the collection
     * @throws RekognitionException if there is an error while interacting with
the Amazon Rekognition service
     */
    public static void addToCollection(RekognitionClient rekClient, String
collectionId, String bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image targetImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();
```

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

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

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 IndexFaces를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun addToCollection(
    collectionIdVal: String?,
    sourceImage: String,
) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        IndexFacesRequest {
            collectionId = collectionIdVal
            image = souImage
            maxFaces = 1
            qualityFilter = QualityFilter.Auto
            detectionAttributes = listOf(Attribute.Default)
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val facesResponse = rekClient.indexFaces(request)

        // Display the results.
        println("Results for the image")
        println("\n Faces indexed:")
        facesResponse.faceRecords?.forEach { faceRecord ->
            println("Face ID: ${faceRecord.face?.faceId}")
            println("Location: ${faceRecord.faceDetail?.boundingBox}")
        }

        println("Faces not indexed:")
        facesResponse.unindexedFaces?.forEach { unindexedFace ->
            println("Location: ${unindexedFace.faceDetail?.boundingBox}")
```

```
            println("Reasons:")

            unindexedFace.reasons?.forEach { reason ->
                println("Reason:  $reason")
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 IndexFaces를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client
```

```
    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def index_faces(self, image, max_faces):
        """
        Finds faces in the specified image, indexes them, and stores them in the
        collection.

        :param image: The image to index.
        :param max_faces: The maximum number of faces to index.
        :return: A tuple. The first element is a list of indexed faces.
                 The second element is a list of faces that couldn't be indexed.
        """
        try:
            response = self.rekognition_client.index_faces(
                CollectionId=self.collection_id,
                Image=image.image,
                ExternalImageId=image.image_name,
                MaxFaces=max_faces,
                DetectionAttributes=["ALL"],
            )
            indexed_faces = [
                RekognitionFace({**face["Face"], **face["FaceDetail"]})
                for face in response["FaceRecords"]
            ]
            unindexed_faces = [
                RekognitionFace(face["FaceDetail"])
                for face in response["UnindexedFaces"]
            ]
            logger.info(
```

```
                "Indexed %s faces in %s. Could not index %s faces.",
                len(indexed_faces),
                image.image_name,
                len(unindexed_faces),
            )
        except ClientError:
            logger.exception("Couldn't index faces in image %s.",
image.image_name)
            raise
        else:
            return indexed_faces, unindexed_faces
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 IndexFaces를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Create S3 object reference for the image
        DATA(lo_s3object) = NEW /aws1/cl_reks3object(
          iv_bucket = iv_s3_bucket
          iv_name = iv_s3_key ).

        " Create image object
        DATA(lo_image) = NEW /aws1/cl_rekimage(
          io_s3object = lo_s3object ).

        " Index faces in the image
        oo_result = lo_rek->indexfaces(
          iv_collectionid = iv_collection_id
          io_image = lo_image
          iv_externalimageid = iv_external_id
          iv_maxfaces = iv_max_faces ).
```

```
      DATA(lt_face_records) = oo_result->get_facerecords( ).
      DATA(lv_indexed_count) = lines( lt_face_records ).
      DATA(lv_msg2) = |{ lv_indexed_count } face(s) indexed successfully.|.
      MESSAGE lv_msg2 TYPE 'I'.
    CATCH /aws1/cx_rekresourcenotfoundex.
      MESSAGE 'Collection not found.' TYPE 'E'.
    CATCH /aws1/cx_rekinvalids3objectex.
      MESSAGE 'Invalid S3 object.' TYPE 'E'.
    CATCH /aws1/cx_rekinvalidparameterex.
      MESSAGE 'Invalid parameter value.' TYPE 'E'.
  ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 IndexFaces를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `ListCollections` 함께 사용

다음 코드 예시는 `ListCollections`의 사용 방법을 보여 줍니다.

자세한 내용은 컬렉션 나열을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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);
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 ListCollections를 참조하세요.

CLI

AWS CLI

사용 가능한 모음을 나열하는 방법

다음 `list-collections` 명령은 AWS 계정에서 사용 가능한 컬렉션을 나열합니다.

```
aws rekognition list-collections
```

출력:

```
{
    "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"
    ]
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 컬렉션 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListCollections를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsRequest;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListCollections {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Listing collections");
        listAllCollections(rekClient);
        rekClient.close();
    }

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

```
                    .maxResults(10)
                    .build();

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListCollections를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun listAllCollections() {
    val request =
        ListCollectionsRequest {
            maxResults = 10
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listCollections(request)
        response.collectionIds?.forEach { resultId ->
            println(resultId)
        }
    }
```

```
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ListCollections를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
 API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client


    def list_collections(self, max_results):
        """
        Lists collections for the current account.

        :param max_results: The maximum number of collections to return.
        :return: The list of collections for the current account.
        """
        try:
            response =
 self.rekognition_client.list_collections(MaxResults=max_results)
            collections = [
```

```
                RekognitionCollection({"CollectionId": col_id},
self.rekognition_client)
                for col_id in response["CollectionIds"]
            ]
        except ClientError:
            logger.exception("Couldn't list collections.")
            raise
        else:
            return collections
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 ListCollections를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_rek->listcollections(
          iv_maxresults = iv_max_results ).

        DATA(lt_collection_ids) = oo_result->get_collectionids( ).
        DATA(lv_coll_count) = lines( lt_collection_ids ).
        DATA(lv_msg7) = |{ lv_coll_count } collection(s) found.|.
        MESSAGE lv_msg7 TYPE 'I'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 ListCollections를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `ListFaces` 함께 사용

다음 코드 예시는 `ListFaces`의 사용 방법을 보여 줍니다.

자세한 내용은 컬렉션에서 얼굴 나열을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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));
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 ListFaces를 참조하세요.

CLI

AWS CLI

모음에 있는 얼굴을 나열하는 방법

다음 `list-faces` 명령은 지정된 컬렉션에 있는 얼굴을 나열합니다.

```
aws rekognition list-faces \
    --collection-id MyCollection
```

출력:

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


자세한 내용은 Amazon Rekognition 개발자 안내서의 컬렉션의 얼굴 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListFaces를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Face;
import software.amazon.awssdk.services.rekognition.model.ListFacesRequest;
import software.amazon.awssdk.services.rekognition.model.ListFacesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

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

                Usage:    <collectionId>

                Where:
                   collectionId - The name of the collection.\s
                """;

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

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

        System.out.println("Faces in collection " + collectionId);
        listFacesCollection(rekClient, collectionId);
        rekClient.close();
    }
```

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

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListFaces를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun listFacesCollection(collectionIdVal: String?) {
    val request =
        ListFacesRequest {
            collectionId = collectionIdVal
            maxResults = 10
        }
```

```
    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listFaces(request)
        response.faces?.forEach { face ->
            println("Confidence level there is a face: ${face.confidence}")
            println("The face Id value is ${face.faceId}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ListFaces를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
```

```
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def list_faces(self, max_results):
        """
        Lists the faces currently indexed in the collection.

        :param max_results: The maximum number of faces to return.
        :return: The list of faces in the collection.
        """
        try:
            response = self.rekognition_client.list_faces(
                CollectionId=self.collection_id, MaxResults=max_results
            )
            faces = [RekognitionFace(face) for face in response["Faces"]]
            logger.info(
                "Found %s faces in collection %s.", len(faces),
self.collection_id
            )
        except ClientError:
            logger.exception(
                "Couldn't list faces in collection %s.", self.collection_id
            )
            raise
        else:
            return faces
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListFaces를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
TRY.
    oo_result = lo_rek->listfaces(
      iv_collectionid = iv_collection_id
      iv_maxresults = iv_max_results ).

    DATA(lt_faces) = oo_result->get_faces( ).
    DATA(lv_face_count2) = lines( lt_faces ).
    DATA(lv_msg3) = |{ lv_face_count2 } face(s) found in collection.|.
    MESSAGE lv_msg3 TYPE 'I'.
  CATCH /aws1/cx_rekresourcenotfoundex.
    MESSAGE 'Collection not found.' TYPE 'E'.
  CATCH /aws1/cx_rekinvalidparameterex.
    MESSAGE 'Invalid parameter value.' TYPE 'E'.
ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 ListFaces를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `RecognizeCelebrities` 함께 사용

다음 코드 예시는 `RecognizeCelebrities`의 사용 방법을 보여 줍니다.

자세한 내용은 이미지에서 유명인 인식을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    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.");
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 RecognizeCelebrities를 참조하세요.

CLI

AWS CLI

이미지에서 유명 인사를 인식하는 방법

다음 `recognize-celebrities` 명령은 Amazon S3 버킷에 저장된 지정된 이미지에서 유명인을 인식합니다.

```
aws rekognition recognize-celebrities \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=moviestars.jpg}"
```

출력:

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

자세한 내용은 Amazon Rekognition 개발자 안내서의 이미지에서 유명인 인식을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 RecognizeCelebrities를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

import software.amazon.awssdk.services.rekognition.model.*;

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

```
public class RecognizeCelebrities {
    public static void main(String[] args) {
        final String usage = """
                Usage:   <bucketName> <sourceImage>

                Where:
                   bucketName - The name of the S3 bucket where the images are
 stored.
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

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

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

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

    /**
     * Recognizes all celebrities in an image stored in an Amazon S3 bucket.
     *
     * @param rekClient    the Amazon Rekognition client used to perform the
 celebrity recognition operation
     * @param bucketName   the name of the Amazon S3 bucket where the source
 image is stored
     * @param sourceImage  the name of the source image file stored in the Amazon
 S3 bucket
     */
    public static void recognizeAllCelebrities(RekognitionClient rekClient,
 String bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                .bucket(bucketName)
```

```
                .name(sourceImage)
                .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

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

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

                System.out.println("Further information (if available):");
                for (String url : celebrity.urls()) {
                    System.out.println(url);
                }
                System.out.println();
            }
            System.out.println(result.unrecognizedFaces().size() + " face(s) were
 unrecognized.");

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 RecognizeCelebrities를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun recognizeAllCelebrities(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        RecognizeCelebritiesRequest {
            image = souImage
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.recognizeCelebrities(request)
        response.celebrityFaces?.forEach { celebrity ->
            println("Celebrity recognized: ${celebrity.name}")
            println("Celebrity ID:${celebrity.id}")
            println("Further information (if available):")
            celebrity.urls?.forEach { url ->
                println(url)
            }
        }
        println("${response.unrecognizedFaces?.size} face(s) were unrecognized.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 RecognizeCelebrities를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def recognize_celebrities(self):
        """
        Detects celebrities in the image.

        :return: A tuple. The first element is the list of celebrities found in
                 the image. The second element is the list of faces that were
                 detected but did not match any known celebrities.
        """
        try:
            response =
 self.rekognition_client.recognize_celebrities(Image=self.image)
            celebrities = [
```

```
                RekognitionCelebrity(celeb) for celeb in
response["CelebrityFaces"]
            ]
            other_faces = [
                RekognitionFace(face) for face in response["UnrecognizedFaces"]
            ]
            logger.info(
                "Found %s celebrities and %s other faces in %s.",
                len(celebrities),
                len(other_faces),
                self.image_name,
            )
        except ClientError:
            logger.exception("Couldn't detect celebrities in %s.",
self.image_name)
            raise
        else:
            return celebrities, other_faces
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 RecognizeCelebrities를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Create S3 object reference for the image
        DATA(lo_s3object) = NEW /aws1/cl_reks3object(
          iv_bucket = iv_s3_bucket
          iv_name = iv_s3_key ).
```

```
        " Create image object
        DATA(lo_image) = NEW /aws1/cl_rekimage(
          io_s3object = lo_s3object ).

        " Recognize celebrities
        oo_result = lo_rek->recognizecelebrities(
          io_image = lo_image ).

        DATA(lt_celebrity_faces) = oo_result->get_celebrityfaces( ).
        DATA(lv_celeb_count) = lines( lt_celebrity_faces ).
        DATA(lv_msg12) = |{ lv_celeb_count } celebrity/celebrities recognized.|.
        MESSAGE lv_msg12 TYPE 'I'.
      CATCH /aws1/cx_rekinvalids3objectex.
        MESSAGE 'Invalid S3 object.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 RecognizeCelebrities를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `SearchFaces` 함께 사용

다음 코드 예시는 SearchFaces의 사용 방법을 보여 줍니다.

자세한 내용은 얼굴(얼굴 ID) 검색을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    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}");
            });
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 SearchFaces를 참조하세요.

CLI

AWS CLI

모음에서 얼굴 ID와 일치하는 얼굴을 검색하는 방법

다음 `search-faces` 명령은 컬렉션에서 지정된 얼굴 ID와 일치하는 얼굴을 검색합니다.

```
aws rekognition search-faces \
    --face-id 8d3cfc70-4ba8-4b36-9644-90fba29c2dac \
    --collection-id MyCollection
```

출력:

```
{
    "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
        }
    ]
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 얼굴 ID를 사용하여 얼굴 검색을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 SearchFaces를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
```

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

                Usage:    <collectionId> <sourceImage>

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

                """;

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

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

        System.out.println("Searching for a face in a collections");
        searchFaceInCollection(rekClient, collectionId, sourceImage);
        rekClient.close();
    }

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

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

            SearchFacesByImageRequest facesByImageRequest =
 SearchFacesByImageRequest.builder()
                    .image(souImage)
                    .maxFaces(10)
                    .faceMatchThreshold(70F)
                    .collectionId(collectionId)
                    .build();

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 SearchFaces를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def search_faces(self, face_id, threshold, max_faces):
        """
        Searches for faces in the collection that match another face from the
        collection.

        :param face_id: The ID of the face in the collection to search for.
```

```
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: The list of matching faces found in the collection. This list
does
                 not contain the face specified by `face_id`.
        """
        try:
            response = self.rekognition_client.search_faces(
                CollectionId=self.collection_id,
                FaceId=face_id,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            faces = [RekognitionFace(face["Face"]) for face in
response["FaceMatches"]]
            logger.info(
                "Found %s faces in %s that match %s.",
                len(faces),
                self.collection_id,
                face_id,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                face_id,
            )
            raise
        else:
            return faces
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 SearchFaces를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_rek->searchfaces(
          iv_collectionid = iv_collection_id
          iv_faceid = iv_face_id
          iv_facematchthreshold = iv_threshold
          iv_maxfaces = iv_max_faces ).

        DATA(lt_face_matches) = oo_result->get_facematches( ).
        DATA(lv_match_count2) = lines( lt_face_matches ).
        DATA(lv_msg5) = |Face search completed: { lv_match_count2 } match(es)
found.|.
        MESSAGE lv_msg5 TYPE 'I'.
      CATCH /aws1/cx_rekresourcenotfoundex.
        MESSAGE 'Collection or face not found.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 SearchFaces를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **SearchFacesByImage** 함께 사용

다음 코드 예시는 SearchFacesByImage의 사용 방법을 보여 줍니다.

자세한 내용은 얼굴 검색(이미지)을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to search for images matching those
/// in a collection.
/// </summary>
public class SearchFacesMatchingImage
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string bucket = "amzn-s3-demo-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        var image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var searchFacesByImageRequest = new SearchFacesByImageRequest()
        {
            CollectionId = collectionId,
```

```
                Image = image,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesByImageResponse searchFacesByImageResponse = await
rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

            Console.WriteLine("Faces matching largest face in image from " +
photo);
            searchFacesByImageResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
            });
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 SearchFacesByImage를 참조하세요.

CLI

AWS CLI

이미지에서 가장 큰 얼굴과 일치하는 얼굴을 모음에서 검색하는 방법

다음 `search-faces-by-image` 명령은 지정된 이미지에서 가장 큰 얼굴과 일치하는 얼굴을 컬렉션에서 검색합니다.

```
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"
}
```

자세한 내용은 Amazon Rekognition 개발자 안내서의 이미지를 사용하여 얼굴 검색을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 SearchFacesByImage를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.SearchFacesRequest;
import software.amazon.awssdk.services.rekognition.model.SearchFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

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

```
 */
public class SearchFaceMatchingIdCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <sourceImage>

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

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

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Searching for a face in a collections");
        searchFacebyId(rekClient, collectionId, faceId);
        rekClient.close();
    }

    public static void searchFacebyId(RekognitionClient rekClient, String
 collectionId, String faceId) {
        try {
            SearchFacesRequest searchFacesRequest = SearchFacesRequest.builder()
                    .collectionId(collectionId)
                    .faceId(faceId)
                    .faceMatchThreshold(70F)
                    .maxFaces(2)
                    .build();

            SearchFacesResponse imageResponse =
 rekClient.searchFaces(searchFacesRequest);
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
```

```
            for (FaceMatch face : faceImageMatches) {
                System.out.println("The similarity level is  " +
 face.similarity());
                System.out.println();
            }

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 SearchFacesByImage를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
```

```
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def search_faces_by_image(self, image, threshold, max_faces):
        """
        Searches for faces in the collection that match the largest face in the
        reference image.

        :param image: The image that contains the reference face to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: A tuple. The first element is the face found in the reference
image.
                 The second element is the list of matching faces found in the
                 collection.
        """
        try:
            response = self.rekognition_client.search_faces_by_image(
                CollectionId=self.collection_id,
                Image=image.image,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            image_face = RekognitionFace(
```

```
                {
                    "BoundingBox": response["SearchedFaceBoundingBox"],
                    "Confidence": response["SearchedFaceConfidence"],
                }
            )
            collection_faces = [
                RekognitionFace(face["Face"]) for face in response["FaceMatches"]
            ]
            logger.info(
                "Found %s faces in the collection that match the largest "
                "face in %s.",
                len(collection_faces),
                image.image_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                image.image_name,
            )
            raise
        else:
            return image_face, collection_faces
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 SearchFacesByImage를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Create S3 object reference for the image
```

```
        DATA(lo_s3object) = NEW /aws1/cl_reks3object(
          iv_bucket = iv_s3_bucket
          iv_name = iv_s3_key ).

        " Create image object
        DATA(lo_image) = NEW /aws1/cl_rekimage(
          io_s3object = lo_s3object ).

        " Search for matching faces
        oo_result = lo_rek->searchfacesbyimage(
          iv_collectionid = iv_collection_id
          io_image = lo_image
          iv_facematchthreshold = iv_threshold
          iv_maxfaces = iv_max_faces ).

        DATA(lt_face_matches) = oo_result->get_facematches( ).
        DATA(lv_match_count) = lines( lt_face_matches ).
        DATA(lv_msg4) = |Face search completed: { lv_match_count } match(es)
found.|.
        MESSAGE lv_msg4 TYPE 'I'.
      CATCH /aws1/cx_rekresourcenotfoundex.
        MESSAGE 'Collection not found.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalids3objectex.
        MESSAGE 'Invalid S3 object.' TYPE 'E'.
      CATCH /aws1/cx_rekinvalidparameterex.
        MESSAGE 'Invalid parameter value.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 SDK for SAP ABAP API 참조의 SearchFacesByImage를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDKs를 사용한 Amazon Rekognition 시나리오

다음 코드 예제에서는 Amazon Rekognition AWS SDKs에서 일반적인 시나리오를 구현하는 방법을 보여줍니다. 이러한 시나리오에서는 Amazon Rekognition 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여줍니다. 각 시나리오에는 전체 소스 코드에 대한 링크가 포함되어 있습니다. 여기에서 코드를 설정 및 실행하는 방법에 대한 지침을 찾을 수 있습니다.

시나리오는 컨텍스트에 맞는 서비스 작업을 이해하는 데 도움이 되도록 중급 수준의 경험을 대상으로 합니다.

예시

- AWS SDK를 사용하여 Amazon Rekognition 컬렉션을 빌드하고 해당 컬렉션에서 얼굴 찾기
- 사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기
- AWS SDK를 사용하여 Amazon Rekognition으로 이미지에서 PPE 감지
- AWS SDK를 사용하여 Amazon Rekognition을 사용하여 이미지의 요소 감지 및 표시
- AWS SDK를 사용하여 이미지에서 얼굴 감지
- Amazon Rekognition 및 AWS SDK를 사용하여 비디오에서 정보 감지
- AWS SDK를 사용하여 Amazon Rekognition으로 이미지의 객체 감지
- AWS SDK를 사용하여 Amazon Rekognition을 사용하여 비디오에서 사람 및 객체 감지
- AWS SDK를 사용하여 EXIF 및 기타 이미지 정보 저장

AWS SDK를 사용하여 Amazon Rekognition 컬렉션을 빌드하고 해당 컬렉션에서 얼굴 찾기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon Rekognition 컬렉션을 생성합니다.
- 컬렉션에 이미지를 추가하고 컬렉션에서 얼굴을 감지합니다.
- 컬렉션에서 참조 이미지와 일치하는 얼굴을 검색합니다.
- 컬렉션을 삭제합니다.

자세한 내용은 컬렉션에서 얼굴 검색을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon Rekognition 함수를 래핑하는 클래스를 생성합니다.

```
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
```

래퍼 클래스를 사용하여 이미지 세트에서 얼굴 컬렉션을 만든 다음 컬렉션에서 얼굴을 검색합니다.

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Rekognition face collection demo!")
    print("-" * 88)

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

    rekognition_client = boto3.client("rekognition")
    images = [
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128316.jpg",
            rekognition_client,
            image_name="sitting",
        ),
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128317.jpg",
            rekognition_client,
            image_name="hopping",
        ),
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128318.jpg",
            rekognition_client,
            image_name="biking",
        ),
    ]

    collection_mgr = RekognitionCollectionManager(rekognition_client)
    collection = collection_mgr.create_collection("doc-example-collection-demo")
    print(f"Created collection {collection.collection_id}:")
    pprint(collection.describe_collection())

    print("Indexing faces from three images:")
    for image in images:
        collection.index_faces(image, 10)
    print("Listing faces in collection:")
    faces = collection.list_faces(10)
    for face in faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")
```

```
    print(
        f"Searching for faces in the collection that match the first face in the
"
        f"list (Face ID: {faces[0].face_id}."
    )
    found_faces = collection.search_faces(faces[0].face_id, 80, 10)
    print(f"Found {len(found_faces)} matching faces.")
    for face in found_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    print(
        f"Searching for faces in the collection that match the largest face in "
        f"{images[0].image_name}."
    )
    image_face, match_faces = collection.search_faces_by_image(images[0], 80, 10)
    print(f"The largest face in {images[0].image_name} is:")
    pprint(image_face.to_dict())
    print(f"Found {len(match_faces)} matching faces.")
    for face in match_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    collection.delete_collection()
    print("Thanks for watching!")
    print("-" * 88)
```

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기

다음 코드 예제는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여 줍니다.

.NET

SDK for .NET

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK for C++

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK for Java 2.x

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK for Kotlin

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK for PHP

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK for Rust

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon Rekognition으로 이미지에서 PPE 감지

다음 코드 예제에서는 Amazon Rekognition을 사용하여 이미지에서 개인 보호 장비(PPE)를 감지하는 앱을 구축하는 방법을 보여줍니다.

Java

SDK for Java 2.x

개인 보호 장비로 이미지를 감지하는 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3

- Amazon SES

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon Rekognition을 사용하여 이미지의 요소 감지 및 표시

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon Rekognition을 사용하여 이미지에서 요소를 감지하고 표시합니다.
- 이미지를 표시하고 감지된 요소 주위에 경계 상자를 그립니다.

자세한 내용은 경계 상자 표시를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon Rekognition 함수를 래핑하는 클래스를 생성합니다.

```
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
```

경계 상자와 다각형을 그리는 도우미 함수를 생성합니다.

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

logger = logging.getLogger(__name__)


def show_bounding_boxes(image_bytes, box_sets, colors):
    """
    Draws bounding boxes on an image and shows it with the default image viewer.

    :param image_bytes: The image to draw, as bytes.
    :param box_sets: A list of lists of bounding boxes to draw on the image.
    :param colors: A list of colors to use to draw the bounding boxes.
    """
    image = Image.open(io.BytesIO(image_bytes))
    draw = ImageDraw.Draw(image)
    for boxes, color in zip(box_sets, colors):
        for box in boxes:
            left = image.width * box["Left"]
            top = image.height * box["Top"]
            right = (image.width * box["Width"]) + left
            bottom = (image.height * box["Height"]) + top
            draw.rectangle([left, top, right, bottom], outline=color, width=3)
    image.show()



def show_polygons(image_bytes, polygons, color):
    """
    Draws polygons on an image and shows it with the default image viewer.

    :param image_bytes: The image to draw, as bytes.
    :param polygons: The list of polygons to draw on the image.
    :param color: The color to use to draw the polygons.
    """
    image = Image.open(io.BytesIO(image_bytes))
```

```
    draw = ImageDraw.Draw(image)
    for polygon in polygons:
        draw.polygon(
            [
                (image.width * point["X"], image.height * point["Y"])
                for point in polygon
            ],
            outline=color,
        )
    image.show()
```

Amazon Rekognition에서 반환한 객체를 파싱하기 위한 클래스를 생성합니다.

```
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
```

래퍼 클래스를 사용하여 이미지에서 요소를 감지하고 해당 요소의 경계 상자를 표시합니다. 이 예제에 사용된 이미지는 지침 및 추가 코드와 함께 GitHub에서 찾을 수 있습니다.

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Rekognition image detection demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    rekognition_client = boto3.client("rekognition")
    street_scene_file_name = ".media/pexels-kaique-rocha-109919.jpg"
    celebrity_file_name = ".media/pexels-pixabay-53370.jpg"
    one_girl_url = "https://dhei5unw3vrsx.cloudfront.net/images/
source3_resized.jpg"
    three_girls_url = "https://dhei5unw3vrsx.cloudfront.net/images/
target3_resized.jpg"
    swimwear_object = boto3.resource("s3").Object(
        "console-sample-images-pdx", "yoga_swimwear.jpg"
    )
    book_file_name = ".media/pexels-christina-morillo-1181671.jpg"

    street_scene_image = RekognitionImage.from_file(
        street_scene_file_name, rekognition_client
    )
    print(f"Detecting faces in {street_scene_image.image_name}...")
    faces = street_scene_image.detect_faces()
    print(f"Found {len(faces)} faces, here are the first three.")
    for face in faces[:3]:
        pprint(face.to_dict())
    show_bounding_boxes(
        street_scene_image.image["Bytes"],
        [[face.bounding_box for face in faces]],
        ["aqua"],
```

```
    )
    input("Press Enter to continue.")

    print(f"Detecting labels in {street_scene_image.image_name}...")
    labels = street_scene_image.detect_labels(100)
    print(f"Found {len(labels)} labels.")
    for label in labels:
        pprint(label.to_dict())
    names = []
    box_sets = []
    colors = ["aqua", "red", "white", "blue", "yellow", "green"]
    for label in labels:
        if label.instances:
            names.append(label.name)
            box_sets.append([inst["BoundingBox"] for inst in label.instances])
    print(f"Showing bounding boxes for {names} in {colors[:len(names)]}.")
    show_bounding_boxes(
        street_scene_image.image["Bytes"], box_sets, colors[: len(names)]
    )
    input("Press Enter to continue.")

    celebrity_image = RekognitionImage.from_file(
        celebrity_file_name, rekognition_client
    )
    print(f"Detecting celebrities in {celebrity_image.image_name}...")
    celebs, others = celebrity_image.recognize_celebrities()
    print(f"Found {len(celebs)} celebrities.")
    for celeb in celebs:
        pprint(celeb.to_dict())
    show_bounding_boxes(
        celebrity_image.image["Bytes"],
        [[celeb.face.bounding_box for celeb in celebs]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    girl_image_response = requests.get(one_girl_url)
    girl_image = RekognitionImage(
        {"Bytes": girl_image_response.content}, "one-girl", rekognition_client
    )
    group_image_response = requests.get(three_girls_url)
    group_image = RekognitionImage(
        {"Bytes": group_image_response.content}, "three-girls",
rekognition_client
```

```
    )
    print("Comparing reference face to group of faces...")
    matches, unmatches = girl_image.compare_faces(group_image, 80)
    print(f"Found {len(matches)} face matching the reference face.")
    show_bounding_boxes(
        group_image.image["Bytes"],
        [[match.bounding_box for match in matches]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    swimwear_image = RekognitionImage.from_bucket(swimwear_object,
rekognition_client)
    print(f"Detecting suggestive content in {swimwear_object.key}...")
    labels = swimwear_image.detect_moderation_labels()
    print(f"Found {len(labels)} moderation labels.")
    for label in labels:
        pprint(label.to_dict())
    input("Press Enter to continue.")

    book_image = RekognitionImage.from_file(book_file_name, rekognition_client)
    print(f"Detecting text in {book_image.image_name}...")
    texts = book_image.detect_text()
    print(f"Found {len(texts)} text instances. Here are the first seven:")
    for text in texts[:7]:
        pprint(text.to_dict())
    show_polygons(
        book_image.image["Bytes"], [text.geometry["Polygon"] for text in texts],
"aqua"
    )

    print("Thanks for watching!")
    print("-" * 88)
```

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 이미지에서 얼굴 감지

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon S3 버킷에 이미지를 저장합니다.
- Amazon Rekognition을 사용하여 연령대, 성별, 감정(예제: 웃음) 등의 얼굴 세부 정보를 감지합니다.
- 이러한 세부 정보를 표시합니다.

Rust

SDK for Rust

uploads 접두사를 사용하여 Amazon S3 버킷에 이미지를 저장하고, Amazon Rekognition을 사용하여 연령대, 성별, 감정(예제: 웃음) 등의 얼굴 세부 정보를 감지한 후 이러한 세부 정보를 표시합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon Rekognition 및 AWS SDK를 사용하여 비디오에서 정보 감지

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon Rekognition 작업을 시작하여 동영상에서 사람, 사물, 텍스트와 같은 요소를 탐지합니다.
- 작업이 완료될 때까지 작업 상태를 확인하세요.
- 각 작업에서 감지한 요소 목록을 출력합니다.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon S3 버킷에 있는 동영상에서 유명인사의 결과를 가져옵니다.

```
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);
        }
    }
}
```

레이블 감지 작업을 통해 동영상의 레이블을 감지합니다.

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
```

```
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;

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

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

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

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

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
```

```
        String roleArn = args[4];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

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

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

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

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

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

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

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

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

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

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

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

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

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

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {
        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
                .queueUrl(queueUrl)
                .build();

        try {
```

```
            messages = sqs.receiveMessage(messageRequest).messages();

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

                    // Get the status and job id from the notification
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found in JSON is " + operationJobId);

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

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

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

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

        } catch (RekognitionException e) {
            e.getMessage();
```

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

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

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

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

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

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

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

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

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

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

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

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

Amazon S3 버킷에 저장된 동영상에서 얼굴을 감지합니다.

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;

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

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

```
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

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

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

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
        String roleArn = args[4];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

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

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

```
    }

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

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

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

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

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

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

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

                if (status.compareTo("SUCCEEDED") == 0)
```

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

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

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

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

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {
        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
                .queueUrl(queueUrl)
                .build();

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

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

                    // Get the status and job id from the notification
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                            .queueUrl(queueUrl)
```

```
                            .build();

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Amazon S3 버킷에 저장된 동영상에서 부적절하거나 불쾌감을 주는 콘텐츠를 감지합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import
 software.amazon.awssdk.services.rekognition.model.ContentModerationDetection;
import java.util.List;
```

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

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

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

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
```

```
                .roleArn(roleArn)
                .build();

        startModerationDetection(rekClient, channel, bucket, video);
        getModResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

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

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

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

            StartContentModerationRequest modDetectionRequest =
StartContentModerationRequest.builder()
                    .jobTag("Moderation")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartContentModerationResponse startModDetectionResult = rekClient
                    .startContentModeration(modDetectionRequest);
            startJobId = startModDetectionResult.jobId();

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

    public static void getModResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
```

```
            GetContentModerationResponse modDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

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

                GetContentModerationRequest modRequest =
GetContentModerationRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    modDetectionResponse =
rekClient.getContentModeration(modRequest);
                    status = modDetectionResponse.jobStatusAsString();

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

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                VideoMetadata videoMetaData =
modDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");
```

```
                List<ContentModerationDetection> mods =
 modDetectionResponse.moderationLabels();
                for (ContentModerationDetection mod : mods) {
                    long seconds = mod.timestamp() / 1000;
                    System.out.print("Mod label: " + seconds + " ");
                    System.out.println(mod.moderationLabel().toString());
                    System.out.println();
                }

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

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

Amazon S3 버킷에 저장된 동영상에서 기술적 큐 세그먼트와 샷 감지 세그먼트를 감지합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartShotDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartTechnicalCueDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionFilters;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
```

```
import software.amazon.awssdk.services.rekognition.model.SegmentDetection;
import software.amazon.awssdk.services.rekognition.model.TechnicalCueSegment;
import software.amazon.awssdk.services.rekognition.model.ShotSegment;
import software.amazon.awssdk.services.rekognition.model.SegmentType;
import software.amazon.awssdk.services.sqs.SqsClient;
import java.util.List;

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

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

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

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
```

```
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

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

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

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

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

            StartShotDetectionFilter cueDetectionFilter =
StartShotDetectionFilter.builder()
                    .minSegmentConfidence(60F)
                    .build();

            StartTechnicalCueDetectionFilter technicalCueDetectionFilter =
StartTechnicalCueDetectionFilter.builder()
                    .minSegmentConfidence(60F)
                    .build();
```

```
            StartSegmentDetectionFilters filters =
StartSegmentDetectionFilters.builder()
                    .shotFilter(cueDetectionFilter)
                    .technicalCueFilter(technicalCueDetectionFilter)
                    .build();

            StartSegmentDetectionRequest segDetectionRequest =
StartSegmentDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .segmentTypes(SegmentType.TECHNICAL_CUE, SegmentType.SHOT)
                    .video(vidOb)
                    .filters(filters)
                    .build();

            StartSegmentDetectionResponse segDetectionResponse =
rekClient.startSegmentDetection(segDetectionRequest);
            startJobId = segDetectionResponse.jobId();

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

    public static void getSegmentResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetSegmentDetectionResponse segDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

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

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

```
                // Wait until the job succeeds.
                while (!finished) {
                    segDetectionResponse =
rekClient.getSegmentDetection(recognitionRequest);
                    status = segDetectionResponse.jobStatusAsString();

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

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                List<VideoMetadata> videoMetaData =
segDetectionResponse.videoMetadata();
                for (VideoMetadata metaData : videoMetaData) {
                    System.out.println("Format: " + metaData.format());
                    System.out.println("Codec: " + metaData.codec());
                    System.out.println("Duration: " + metaData.durationMillis());
                    System.out.println("FrameRate: " + metaData.frameRate());
                    System.out.println("Job");
                }

                List<SegmentDetection> detectedSegments =
segDetectionResponse.segments();
                for (SegmentDetection detectedSegment : detectedSegments) {
                    String type = detectedSegment.type().toString();
                    if (type.contains(SegmentType.TECHNICAL_CUE.toString())) {
                        System.out.println("Technical Cue");
                        TechnicalCueSegment segmentCue =
detectedSegment.technicalCueSegment();
                        System.out.println("\tType: " + segmentCue.type());
                        System.out.println("\tConfidence: " +
segmentCue.confidence().toString());
                    }

                    if (type.contains(SegmentType.SHOT.toString())) {
                        System.out.println("Shot");
```

```
                        ShotSegment segmentShot = detectedSegment.shotSegment();
                        System.out.println("\tIndex " + segmentShot.index());
                        System.out.println("\tConfidence: " +
 segmentShot.confidence().toString());
                    }

                    long seconds = detectedSegment.durationMillis();
                    System.out.println("\tDuration : " + seconds + "
 milliseconds");
                    System.out.println("\tStart time code: " +
 detectedSegment.startTimecodeSMPTE());
                    System.out.println("\tEnd time code: " +
 detectedSegment.endTimecodeSMPTE());
                    System.out.println("\tDuration time code: " +
 detectedSegment.durationSMPTE());
                    System.out.println();
                }

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

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

Amazon S3 버킷에 저장된 동영상에서 텍스트를 감지합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
```

```
import software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;

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

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

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

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
```

```
                .region(region)
                .build();

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

        startTextLabels(rekClient, channel, bucket, video);
        getTextResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

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

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

            StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

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

```
   public static void getTextResults(RekognitionClient rekClient) {
       try {
           String paginationToken = null;
           GetTextDetectionResponse textDetectionResponse = null;
           boolean finished = false;
           String status;
           int yy = 0;

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

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

               // Wait until the job succeeds.
               while (!finished) {
                   textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                   status = textDetectionResponse.jobStatusAsString();

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

               finished = false;

               // Proceed when the job is done - otherwise VideoMetadata is
null.
               VideoMetadata videoMetaData =
textDetectionResponse.videoMetadata();
               System.out.println("Format: " + videoMetaData.format());
               System.out.println("Codec: " + videoMetaData.codec());
```

```
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<TextDetectionResult> labels =
 textDetectionResponse.textDetections();
                for (TextDetectionResult detectedText : labels) {
                    System.out.println("Confidence: " +
 detectedText.textDetection().confidence().toString());
                    System.out.println("Id : " +
 detectedText.textDetection().id());
                    System.out.println("Parent Id: " +
 detectedText.textDetection().parentId());
                    System.out.println("Type: " +
 detectedText.textDetection().type());
                    System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                    System.out.println();
                }

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

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

Amazon S3 버킷에 저장된 동영상에서 사람을 감지합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingResponse;
```

```
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.PersonDetection;
import java.util.List;

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

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

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

        String bucket = args[0];
        String video = args[1];
```

```
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

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

        startPersonLabels(rekClient, channel, bucket, video);
        getPersonDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

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

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

            StartPersonTrackingRequest personTrackingRequest =
StartPersonTrackingRequest.builder()
                    .jobTag("DetectingLabels")
                    .video(vidOb)
                    .notificationChannel(channel)
                    .build();

            StartPersonTrackingResponse labelDetectionResponse =
rekClient.startPersonTracking(personTrackingRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
```

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

    public static void getPersonDetectionResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetPersonTrackingResponse personTrackingResult = null;
            boolean finished = false;
            String status;
            int yy = 0;

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

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

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

                    personTrackingResult =
rekClient.getPersonTracking(recognitionRequest);
                    status = personTrackingResult.jobStatusAsString();

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

                finished = false;

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

```
                VideoMetadata videoMetaData =
 personTrackingResult.videoMetadata();

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

                List<PersonDetection> detectedPersons =
 personTrackingResult.persons();
                for (PersonDetection detectedPerson : detectedPersons) {
                    long seconds = detectedPerson.timestamp() / 1000;
                    System.out.print("Sec: " + seconds + " ");
                    System.out.println("Person Identifier: " +
 detectedPerson.person().index());
                    System.out.println();
                }

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

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

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection
  - GetTextDetection
  - StartCelebrityRecognition
  - StartContentModeration

- StartLabelDetection
- StartPersonTracking
- StartSegmentDetection
- StartTextDetection

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon S3 버킷에 저장된 동영상에서 얼굴을 감지합니다.

```
suspend fun startFaceDetection(
    channelVal: NotificationChannel?,
    bucketVal: String,
    videoVal: String,
) {
    val s3Obj =
        S3Object {
            bucket = bucketVal
            name = videoVal
        }
    val vidOb =
        Video {
            s3Object = s3Obj
        }

    val request =
        StartFaceDetectionRequest {
            jobTag = "Faces"
            faceAttributes = FaceAttributes.All
            notificationChannel = channelVal
            video = vidOb
        }
```

```
    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val startLabelDetectionResult = rekClient.startFaceDetection(request)
        startJobId = startLabelDetectionResult.jobId.toString()
    }
}

suspend fun getFaceResults() {
    var finished = false
    var status: String
    var yy = 0
    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        var response: GetFaceDetectionResponse? = null

        val recognitionRequest =
            GetFaceDetectionRequest {
                jobId = startJobId
                maxResults = 10
            }

        // Wait until the job succeeds.
        while (!finished) {
            response = rekClient.getFaceDetection(recognitionRequest)
            status = response.jobStatus.toString()
            if (status.compareTo("Succeeded") == 0) {
                finished = true
            } else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }

        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = response?.videoMetadata
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
        println("FrameRate: ${videoMetaData?.frameRate}")

        // Show face information.
        response?.faces?.forEach { face ->
            println("Age: ${face.face?.ageRange}")
            println("Face: ${face.face?.beard}")
            println("Eye glasses: ${face?.face?.eyeglasses}")
```

```
            println("Mustache: ${face.face?.mustache}")
            println("Smile: ${face.face?.smile}")
        }
    }
}
```

Amazon S3 버킷에 저장된 동영상에서 부적절하거나 불쾌감을 주는 콘텐츠를 감지합니다.

```
suspend fun startModerationDetection(
    channel: NotificationChannel?,
    bucketVal: String?,
    videoVal: String?,
) {
    val s3Obj =
        S3Object {
            bucket = bucketVal
            name = videoVal
        }
    val vidOb =
        Video {
            s3Object = s3Obj
        }
    val request =
        StartContentModerationRequest {
            jobTag = "Moderation"
            notificationChannel = channel
            video = vidOb
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val startModDetectionResult = rekClient.startContentModeration(request)
        startJobId = startModDetectionResult.jobId.toString()
    }
}

suspend fun getModResults() {
    var finished = false
    var status: String
    var yy = 0
    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        var modDetectionResponse: GetContentModerationResponse? = null
```

```
        val modRequest =
            GetContentModerationRequest {
                jobId = startJobId
                maxResults = 10
            }

        // Wait until the job succeeds.
        while (!finished) {
            modDetectionResponse = rekClient.getContentModeration(modRequest)
            status = modDetectionResponse.jobStatus.toString()
            if (status.compareTo("Succeeded") == 0) {
                finished = true
            } else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }

        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = modDetectionResponse?.videoMetadata
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
        println("FrameRate: ${videoMetaData?.frameRate}")

        modDetectionResponse?.moderationLabels?.forEach { mod ->
            val seconds: Long = mod.timestamp / 1000
            print("Mod label: $seconds ")
            println(mod.moderationLabel)
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 다음 주제를 참조하세요.
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection

- GetTextDetection
- StartCelebrityRecognition
- StartContentModeration
- StartLabelDetection
- StartPersonTracking
- StartSegmentDetection
- StartTextDetection

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon Rekognition으로 이미지의 객체 감지

다음 코드 예제는 Amazon Rekognition을 사용하여 이미지에서 범주별 객체를 감지하는 앱을 구축하는 방법을 보여줍니다.

.NET

SDK for .NET

Amazon Rekognition .NET을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 이미지에서 범주별로 객체를 식별하기 위해 Amazon Rekognition을 사용하여 앱을 만드는 방법을 보여줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

Java

SDK for Java 2.x

Amazon Rekognition을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 이미지에서 범주별로 객체를 식별하기 위해 Amazon Rekognition을 사용하여 앱을 만드는 방법을 보여줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript(v3)

에서 Amazon Rekognition AWS SDK for JavaScript 을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 이미지에서 Amazon Rekognition을 사용하여 범주별로 객체를 식별하는 앱을 생성하는 방법을 보여줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

다음 작업을 수행하는 방법에 대해 알아보세요.

- Amazon Cognito를 사용하여 인증되지 않은 사용자를 만듭니다.
- Amazon Rekognition을 사용하여 객체용 이미지를 분석합니다.
- Amazon SES 이메일 주소를 확인합니다.
- Amazon SES를 사용하여 이메일 알림을 전송합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

Kotlin

SDK for Kotlin

Amazon Rekognition Kotlin API를 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 이미지에서 범주별로 객체를 식별하기 위해 Amazon Rekognition을 사용하여 앱을 만드는 방법을 보여줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

Python

SDK for Python(Boto3)

를 사용하여 다음을 수행할 수 있는 웹 애플리케이션을 AWS SDK for Python (Boto3) 생성하는 방법을 보여줍니다.

- 사진을 Amazon Simple Storage Service(Amazon S3) 버킷에 업로드합니다.
- Amazon Rekognition을 사용하여 사진을 분석하고 레이블을 지정합니다.
- Amazon Simple Email Service(Amazon SES)를 사용하여 이미지 분석에 대한 이메일 보고서를 보냅니다.

이 예제에는 두 가지 주요 구성 요소가 포함되어 있습니다. 바로 JavaScript로 작성되고 React로 빌드된 웹 페이지와 Python으로 작성되고 Flask-RESTful로 빌드된 REST 서비스입니다.

React 웹 페이지를 사용하여 다음을 수행할 수 있습니다.

- S3 버킷에 저장된 이미지 목록을 표시합니다.
- 컴퓨터에서 S3 버킷에 이미지를 업로드합니다.
- 이미지에서 감지된 항목을 식별하는 이미지와 레이블을 표시합니다.
- S3 버킷의 모든 이미지에 대한 보고서를 받고 보고서의 이메일을 보냅니다.

웹 페이지가 REST 서비스를 호출합니다. 서비스가 다음 작업을 수행하기 위해 AWS 에 요청을 전송합니다.

- S3 버킷의 이미지 목록을 가져오고 필터링합니다.
- S3 버킷에 사진을 업로드합니다.
- Amazon Rekognition을 사용하여 개별 사진을 분석하고 사진에서 감지된 항목을 식별하는 레이블 목록을 가져옵니다.
- S3 버킷의 모든 사진을 분석하고 Amazon SES를 사용하여 보고서를 이메일로 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon Rekognition을 사용하여 비디오에서 사람 및 객체 감지

다음 코드 예제에서는 Amazon Rekognition을 사용하여 동영상에서 사람과 객체를 감지하는 방법을 보여줍니다.

Java

SDK for Java 2.x

Amazon Rekognition Java API를 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 동영상에서 얼굴과 객체를 감지하기 위한 앱을 만드는 방법을 보여줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3

- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK for Python(Boto3)

Amazon Rekognition을 사용하여 비동기식 감지 작업을 시작해 동영상의 얼굴, 객체 및 사람을 감지할 수 있습니다. 또한 이 예제에서는 작업이 완료되고 주제에 대한 Amazon Simple Queue Service(Amazon SQS) 대기열을 구독할 때 Amazon Simple Notification Service(Amazon SNS) 주제를 알리도록 Amazon Rekognition을 구성합니다. 대기열이 작업에 대한 메시지를 받으면 작업이 검색되고 결과가 출력됩니다.

이 예제는 GitHub에서 가장 잘 볼 수 있습니다. 전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 EXIF 및 기타 이미지 정보 저장

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져옵니다.
- Amazon S3 버킷에 이미지 파일을 업로드합니다.
- Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(레이블)을 파악합니다.
- EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

Rust

SDK for Rust

JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져오고, 이미지 파일을 Amazon S3 버킷에 업로드하며, Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(Amazon Rekognition의 레이블)을 파악한 후 EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Rekognition 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

API 참조

Amazon Rekognition API 참조는 이제 Amazon Rekognition API 참조에 있습니다.

Amazon Rekognition 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

다음 주제에서 Amazon Rekognition 리소스를 안전하게 보호하는 방법에 대해 알아보세요.

주제

- Amazon Rekognition의 자격 증명 및 액세스 관리
- Amazon Rekognition의 데이터 보호
- Amazon Rekognition에서 Amazon VPC 엔드포인트 사용
- Amazon Rekognition의 규정 준수 확인
- Amazon Rekognition의 복원성
- Amazon Rekognition의 구성 및 취약성 분석
- 교차 서비스 혼동된 대리인 방지
- Amazon Rekognition의 인프라 보안

Amazon Rekognition의 자격 증명 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 어떤 사용자가 Amazon Rekognition 리소스를 사용할 수 있도록 인증(로그인)되고 권한이 부여(권한 있음)될 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- Amazon Rekognition에서 IAM을 사용하는 방법
- AWS Amazon Rekognition에 대한 관리형 정책
- 정책 모범 사례
- Amazon Rekognition 콘솔 사용

- Amazon Rekognition 자격 증명 기반 정책 예제 사용의 예시
- Amazon Rekognition 리소스 기반 정책의 예
- Amazon Rekognition 자격 증명 및 액세스 문제 해결

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조Amazon Rekognition 자격 증명 및 액세스 문제 해결)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(Amazon Rekognition에서 IAM을 사용하는 방법 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(Amazon Rekognition 자격 증명 기반 정책 예제 사용의 예시 참조)

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구하기를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

IAM 역할

IAM 역할은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. 사용자에서 IAM 역할(콘솔)로 전환하거나 또는 API 작업을 호출하여 역할을 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 권한을 정의합니다.

자격 증명 기반 정책 사용

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택 섹션을 참조하세요.

리소스 기반 정책 사용

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 IAM 엔터티의 권한 범위를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은AWS Organizations 사용 설명서의 서비스 제어 정책을 참조하세요.
- 리소스 제어 정책(RCP) – 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon Rekognition에서 IAM을 사용하는 방법

IAM을 사용하여 Amazon Rekognition에 대한 액세스를 관리하기 전에 Amazon Rekognition에서 사용할 수 있는 IAM 기능을 이해해야 합니다. Amazon Rekognition 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

주제

- Amazon Rekognition 자격 증명 기반 정책
- Amazon Rekognition 리소스 기반 정책
- Amazon Rekognition IAM 역할

Amazon Rekognition 자격 증명 기반 정책

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. Amazon Rekognition은 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

작업

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Amazon Rekognition의 정책 작업은 작업 앞에 `rekognition:` 접두사를 사용합니다. 예를 들어 Amazon Rekognition `DetectLabels` API 작업을 사용하여 이미지의 객체, 장면 또는 개념을 감지할 수 있는 권한을 다른 사람에게 부여하려면 `rekognition:DetectLabels` 작업을 정책에 포함시킵니다. 정책 문에는 `Action` 또는 `NotAction` 요소가 포함되어야 합니다. Amazon Rekognition은 이 서비스로 수행할 수 있는 태스크를 설명하는 고유한 작업 세트를 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "rekognition:action1",
      "rekognition:action2"
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "rekognition:Describe*"
```

Amazon Rekognition 작업의 목록을 보려면 IAM 사용 설명서의 Amazon Rekognition에서 정의한 작업을 참조하세요.

리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

ARN 형식에 대한 자세한 내용은 Amazon 리소스 이름(ARNs) 및 AWS 서비스 네임스페이스를 참조하세요.

예를 들어 설명문에서 `MyCollection` 컬렉션을 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/MyCollection"
```

특정 계정에 속하는 모든 인스턴스를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/*"
```

리소스 생성과 같은 일부 Amazon Rekognition 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

Amazon Rekognition 리소스 유형 및 해당 ARN의 목록을 보려면 IAM 사용 설명서의 Amazon Rekognition에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon Rekognition에서 정의한 작업을 참조하세요.

조건 키

Amazon Rekognition은 서비스별 조건 키를 제공하지 않지만, 일부 전역 조건 키 사용을 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

Amazon Rekognition 리소스 기반 정책

Amazon Rekognition은 Custom Labels 모델을 복사하는 작업에 대해 리소스 기반 정책을 지원합니다. 자세한 내용은 Amazon Rekognition 리소스 기반 정책 예제를 참조하세요.

Amazon S3과 같은 다른 서비스도 리소스 기반 권한 정책을 지원합니다. 예를 들어, 정책을 S3 버킷에 연결하여 해당 버킷에 대한 액세스 권한을 관리할 수 있습니다.

Amazon S3 버킷에 저장된 이미지에 액세스하려면 S3 버킷에 있는 객체에 대한 액세스 권한이 있어야 합니다. 이 권한이 있으면 Amazon Rekognition이 S3 버킷에서 이미지를 다운로드할 수 있습니다. 다음 정책 예제에서 사용자는 amzn-s3-demo-bucket3라는 S3 버킷에서 `s3:GetObject` 작업을 수행할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket3/*"
            ]
        }
    ]
}
```

버전 관리를 활성화한 상태로 S3 버킷을 사용하려면 다음 예제와 같이 `s3:GetObjectVersion` 작업을 추가합니다.

JSON

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

```
                "arn:aws:s3:::amzn-s3-demo-bucket3/*"
            ]
        }
    ]
}
```

Amazon Rekognition IAM 역할

IAM 역할은 특정 권한이 있는 AWS 계정 내 엔터티입니다.

Amazon Rekognition에서 임시 보안 인증 사용

임시 보안 인증을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. AssumeRole 또는 GetFederationToken과 같은 AWS STS API 작업을 호출하여 임시 보안 자격 증명을 얻습니다.

Amazon Rekognition은 임시 보안 인증 사용을 지원합니다.

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 서비스가 다른 서비스의 리소스에 액세스하여 사용자를 대신하여 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수 없습니다.

Amazon Rekognition은 서비스 연결 역할을 지원하지 않습니다.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 서비스 역할을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

Amazon Rekognition은 서비스 역할을 지원합니다.

서비스 역할을 사용하면 Amazon Rekognition을 사용하여 다른 서비스를 직접 호출하고 액세스할 수 없는 리소스를 조작하는 보안 문제가 발생할 수 있습니다. 계정 보안을 유지하려면 Amazon Rekognition의 액세스 범위를 사용 중인 리소스로만 제한합니다. 이는 IAM 서비스 역할에 신뢰 정책을 추가하는 것으로 수행할 수 있습니다. 이렇게 하는 방법에 대한 정보는 교차 서비스 혼동된 대리인 방지 단원을 참조하십시오.

Amazon Rekognition에서 IAM 역할 선택

Amazon Rekognition을 구성하여 저장된 비디오를 분석할 때 Amazon Rekognition에서 사용자를 대신하여 Amazon SNS 에 액세스할 수 있도록 역할을 선택해야 합니다. 이전에 서비스 역할 또는 서비스 연결 역할을 생성한 경우 Amazon Rekognition은 선택할 수 있는 역할 목록을 제공합니다. 자세한 내용은 the section called “Amazon Rekognition Video 구성” 단원을 참조하십시오.

예: Amazon S3 버킷의 이미지에 액세스하도록 Amazon Rekognition 구성

다음은 Amazon S3 버킷의 이미지를 분석하기 위해 Amazon Rekognition을 구성하는 방법의 예입니다. Amazon Rekognition을 사용하여 Amazon S3 버킷의 이미지를 분석하려면 다음을 수행해야 합니다.

1. IAM 사용자/역할(클라이언트)에 관련 Amazon Rekognition API 작업(예: DetectLabels, DetectFaces 등)을 직접 호출할 수 있는 권한이 있는지 확인합니다.

   원하는 API 작업을 간접 호출할 수 있는 적절한 권한을 부여하는 ID 기반 정책을 연결합니다. 예를 들어 역할에 `DetectLabels` 및 `DetectFaces`를 직접 호출할 수 있는 권한을 부여하려면 역할에 다음과 같은 정책을 연결합니다.

   JSON

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

2. Amazon Rekognition 서비스는 Amazon S3 버킷에 액세스할 수 있는 권한이 필요합니다. API 직접 호출 시 Amazon Rekognition에 전달해야 하는 IAM 서비스 역할을 생성합니다. 서비스 역할: Amazon Rekognition 서비스 위탁자를 신뢰하고 버킷에 대한 `s3:GetObject` 권한이 있는지 확인합니다.

신뢰 정책은 다음과 같을 수 있습니다.

JSON

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

서비스 역할에 연결된 ID 기반 정책은 다음과 같을 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS Amazon Rekognition에 대한 관리형 정책

사용자, 그룹 및 역할에 권한을 추가하려면 직접 정책을 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 더 쉽습니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하기 위해서는 시간과

전문 지식이 필요합니다. 빠르게 시작하려면 AWS 관리형 정책을 사용할 수 있습니다. 이러한 정책은 일반적인 사용 사례를 다루며 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 서비스는 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스에서 때때로 추가 권한을 AWS 관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새로운 기능이 시작되거나 새 작업을 사용할 수 있을 때 AWS 관리형 정책에 업데이트됩니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않으므로 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한 여러 서비스에 걸쳐 있는 직무에 대한 관리형 정책을 AWS 지원합니다. 예를 들어 ReadOnlyAccess AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스를 제공합니다. 서비스가 새 기능을 시작하면는 새 작업 및 리소스에 대한 읽기 전용 권한을 AWS 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한AWS 관리형 정책을 참조하세요.

AWS 관리형 정책: AmazonRekognitionFullAccess

`AmazonRekognitionFullAccess`는 컬렉션 생성 및 삭제를 포함하여 Amazon Rekognition 리소스에 대한 모든 액세스 권한을 부여합니다.

`AmazonRekognitionFullAccess` 정책을 IAM ID에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

JSON

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

```
    ]
}
```

AWS 관리형 정책: AmazonRekognitionReadOnlyAccess

AmazonRekognitionReadOnlyAccess는 Amazon Rekognition 리소스에 대한 읽기 전용 액세스 권한을 부여합니다.

AmazonRekognitionReadOnlyAccess 정책을 IAM ID에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

JSON

```
{
    "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 관리형 정책: AmazonRekognitionServiceRole

`AmazonRekognitionServiceRole`는 Amazon Rekognition에서 사용자를 대신하여 Amazon Kinesis Data Streams 및 Amazon SNS 서비스를 직접 호출할 수 있도록 합니다.

`AmazonRekognitionServiceRole` 정책을 IAM ID에 연결할 수 있습니다.

이 서비스 역할을 사용한다면 계정 보안 유지를 위해 Amazon Rekognition의 액세스 범위를 사용 중인 리소스로만 제한해야 합니다. 이는 IAM 서비스 역할에 신뢰 정책을 추가하는 것으로 수행할 수 있습니다. 이렇게 하는 방법에 대한 정보는 교차 서비스 혼동된 대리인 방지 단원을 참조하십시오.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

JSON

```
{
    "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 관리형 정책: AmazonRekognitionCustomLabelsFullAccess

이 정책은 Amazon Rekognition Custom Labels 사용자를 위한 것입니다. AmazonRekognitionCustomLabelsFullAccess 정책을 사용하여 사용자에게 Amazon Rekognition Custom Labels API에 대한 전체 액세스 권한과 Amazon Rekognition Custom Labels 콘솔에서 생성한 콘솔 버킷에 대한 전체 액세스 권한을 부여할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

JSON

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:ListAllMyBuckets",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*custom-labels*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CopyProjectVersion",
                "rekognition:CreateProject",
                "rekognition:CreateProjectVersion",
                "rekognition:StartProjectVersion",
                "rekognition:StopProjectVersion",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DeleteProject",
                "rekognition:DeleteProjectVersion",
                "rekognition:TagResource",
                "rekognition:UntagResource",
                "rekognition:ListTagsForResource",
                "rekognition:CreateDataset",
                "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset",
                "rekognition:UpdateDatasetEntries",
                "rekognition:DistributeDatasetEntries",
                "rekognition:DeleteDataset",
                "rekognition:PutProjectPolicy",
                "rekognition:ListProjectPolicies",
                "rekognition:DeleteProjectPolicy"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

AWS 관리형 정책에 대한 Amazon Rekognition 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후부터 Amazon Rekognition의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 Amazon Rekognition 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경	설명	Date
미디어 분석 작업과 관련된 작업이 다음 관리형 정책에 추가되었습니다. • AWS 관리형 정책: AmazonRekognitionReadOnlyAccess	Amazon Rekognition에서 AmazonRekognitionReadOnlyAccess 관리형 정책에 다음 작업이 추가되었습니다. • GetMediaAnalysisJob • ListMediaAnalysisJob	2023년 10월 31일
사용자 관리와 관련된 작업이 다음 관리형 정책에 추가되었습니다. • AWS 관리형 정책: AmazonRekognitionReadOnlyAccess	Amazon Rekognition에서 AmazonRekognitionReadOnlyAccess 관리형 정책에 다음 작업이 추가되었습니다. • ListUsers • SearchUsers • SearchUsersByImage	2023년 6월 12일
ProjectPolicy 및 사용자 지정 레이블 모델 복사에 대한 작업이 다음 관리형 정책에 추가되었습니다.	Amazon Rekognition의 AmazonRekognitionCustomLabelsFullAccess 및 AmazonRek	2022년 7월 21일

변경	설명	Date
• AWS 관리형 정책: AmazonRekognitionFullAccess • AWS 관리형 정책: AmazonRekognitionCustomLabelsFullAccess	ognitionFullAccess 관리형 정책에 다음 작업이 추가되었습니다. • CopyProjectVersion • PutProjectPolicy • ListProjectPolicies • DeleteProjectPolicy	
ProjectPolicy 및 사용자 지정 레이블 모델 복사에 대한 작업이 다음 관리형 정책에 추가되었습니다. • AWS 관리형 정책: AmazonRekognitionReadOnlyAccess	Amazon Rekognition에서 AmazonRekognitionReadOnlyAccess 관리형 정책에 다음 작업이 추가되었습니다. • ListProjectPolicies	2022년 7월 21일

변경	설명	Date
다음 관리형 정책에 대한 데이터 세트 관리 업데이트 • AWS 관리형 정책: AmazonRekognitionReadOnlyAccess • AWS 관리형 정책: AmazonRekognitionFullAccess • AWS 관리형 정책: AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition에서 AmazonRekognitionReadOnlyAccess, AmazonRekognitionFullOnlyAccess, AmazonRekognitionCustomLabelsFullAccess 관리형 정책에 다음과 같은 작업을 추가했습니다. • `CreateDataset` • `ListDatasetEntries` • `ListDatasetLabels` • `DescribeDataset` • `UpdateDatasetEntries` • `DistributeDatasetEntries` • `DeleteDataset`	2021년 11월 1일
AWS 관리형 정책: AmazonRekognitionReadOnlyAccess 및 AWS 관리형 정책: AmazonRekognitionFullAccess 에 대한 태그 지정 업데이트	Amazon Rekognition에서 AmazonRekognitionFullAccess 및 AmazonRekognitionReadOnlyAccess 정책에 새로운 태그 지정 작업을 추가했습니다.	2021년 4월 2일
Amazon Rekognition 변경 사항 추적 시작	Amazon Rekognition은 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2021년 4월 2일

정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 Amazon Rekognition 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Amazon Rekognition 콘솔 사용

Amazon Rekognition Custom Labels 기능을 제외하고 Amazon Rekognition에서는 Amazon Rekognition 콘솔을 사용할 때 추가 권한이 필요하지 않습니다. Amazon Rekognition Custom Labels에 대한 자세한 내용은 5단계: Amazon Rekognition Custom Labels 콘솔 권한 설정을 참조하세요.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

Amazon Rekognition 자격 증명 기반 정책 예제 사용의 예시

기본적으로 사용자 및 역할은 Amazon Rekognition 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI또는 AWS API를 사용하여 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용자 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- Amazon Rekognition Custom Labels 정책 예제
- 예제 1: 사용자에게 리소스에 대한 읽기 전용 액세스 허용
- 예제 2: 사용자에게 리소스에 대한 모든 액세스 권한 허용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

Amazon Rekognition Custom Labels 정책 예제

Amazon Rekognition Custom Labels의 자격 증명 기반 정책을 생성할 수 있습니다. 자세한 내용은 보안을 참조하세요.

예제 1: 사용자에게 리소스에 대한 읽기 전용 액세스 허용

다음 예제에서는 Amazon Rekognition 리소스에 대한 읽기 전용 액세스 권한을 부여합니다.

JSON

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

예제 2: 사용자에게 리소스에 대한 모든 액세스 권한 허용

다음 예제에서는 Amazon Rekognition 리소스에 대한 모든 액세스 권한을 부여합니다.

JSON

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:*"
            ],
            "Resource": "*"
        }
    ]
}
```

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "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 리소스 기반 정책의 예

Amazon Rekognition Custom Labels는 프로젝트 정책이라고 하는 리소스 기반 정책을 사용하여 모델 버전의 복사 권한을 관리합니다.

프로젝트 정책은 소스 프로젝트에서 대상 프로젝트로 모델 버전을 복사할 수 있는 권한을 허용하거나 거부합니다. 대상 프로젝트가 다른 AWS 계정에 있거나 계정 내 액세스를 제한하려는 경우 프로젝트 정책이 필요합니다 AWS . 예를 들어 특정 IAM 역할에 대한 복사 권한을 거부할 수 있습니다. 자세한 내용은 모델 복사를 참조하세요.

모델 버전 복사 권한 부여

다음 예제는 보안 주체인 `arn:aws:iam::123456789012:role/Admin`에게 모델 버전인 `arn:aws:rekognition:us-east-1:123456789012:project/my_project/version/test_1/1627045542080` 항목을 복사할 수 있도록 허용합니다.

JSON

```
{
  "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 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 Amazon Rekognition 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

주제

- Amazon Rekognition에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 관리자인데, 다른 사용자가 Amazon Rekognition에 액세스할 수 있게 허용하기를 원함
- 내 AWS 계정 외부의 사람이 내 Amazon Rekognition 리소스에 액세스하도록 허용하고 싶습니다.

Amazon Rekognition에서 작업을 수행할 권한이 없음

에서 작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 문의하여 지원을 받아야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

다음 예 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 *##*에 대한 세부 정보를 보려고 하지만 `rekognition:`*`GetWidget`* 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rekognition:GetWidget on resource: my-example-widget
```

이 경우, Mateo는 *`my-example-widget`* 작업을 사용하여 `rekognition:`*`GetWidget`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon Rekognition에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예제 오류는 `marymajor`이라는 IAM 사용자가 콘솔을 사용하여 Amazon Rekognition에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 `iam:PassRole`작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

관리자인데, 다른 사용자가 Amazon Rekognition에 액세스할 수 있게 허용하기를 원함

다른 사용자가 Amazon Rekognition에 액세스하도록 허용하려면 액세스가 필요한 사람 또는 애플리케이션에 권한을 부여해야 합니다. AWS IAM Identity Center 를 사용하여 사용자 및 애플리케이션을 관리하는 경우 사용자 또는 그룹에 권한 세트를 할당하여 액세스 수준을 정의합니다. 권한 세트는 IAM 정책을 자동으로 생성하고 사용자 또는 애플리케이션과 연결된 IAM 역할에 할당합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 권한 세트를 참조하세요.

IAM Identity Center를 사용하지 않는 경우 액세스가 필요한 사용자 또는 애플리케이션에 대한 IAM 엔터티(사용자 또는 역할)를 생성해야 합니다. 그런 다음 Amazon Rekognition에서 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다. 권한이 부여되면 사용자 또는 애플리케이션 개발자에게 자격 증명을 제공합니다. 이들은 이 자격 증명을 사용하여 AWS에 액세스합니다. IAM 사용자, 그룹, 정책 및 권한 생성에 대해 자세히 알아보려면 IAM 사용자 설명서의 IAM 자격 증명과 IAM의 권한 및 정책을 참조하세요.

내 AWS 계정 외부의 사람이 내 Amazon Rekognition 리소스에 액세스하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- Amazon Rekognition에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon Rekognition에서 IAM을 사용하는 방법 섹션을 참조하세요.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사가 AWS 계정 소유한에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

Amazon Rekognition의 데이터 보호

AWS 공동 책임 모델은 Amazon Rekognition의 데이터 보호에 적용됩니다. 이 모델에서 설명하는 것처럼 AWS은 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정 자격 증명을 보호하고 AWS IAM Identity Center 또는 AWS Identity and Access Management(IAM)를 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신하세요. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정하세요. AWS 활동 캡처에 CloudTrail 추적을 사용하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용하세요.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-3 검증된 암호화 모듈이 필요한 경우, FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 Rekognition 또는 기타 AWS 서비스에서 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

데이터 암호화

다음 정보는 Amazon Rekognition에서 데이터 암호화를 사용하여 데이터를 보호하는 방법을 설명합니다.

저장 시 암호화

Amazon Rekognition Image

이미지

Amazon Rekognition API 작업으로 전달된 이미지는 저장되어 서비스를 개선하는 데 사용될 수 있습니다. 단, AI 서비스 옵트아웃 정책 페이지를 방문하여 거기에 설명된 프로세스에 따라 옵트아웃한 경우는 예외입니다. 저장된 이미지는 AWS Key Management Service(SSE-KMS)를 사용하여 저장 시 암호화(Amazon S3)됩니다.

컬렉션

컬렉션에 정보를 저장하는 얼굴 비교 작업의 경우 기본 감지 알고리즘은 먼저 입력 이미지에서 얼굴을 감지하고 각 얼굴에 대한 벡터를 추출한 다음 얼굴 벡터를 컬렉션에 저장합니다. Amazon Rekognition은 얼굴 비교를 수행할 때 이러한 얼굴 벡터를 사용합니다. 얼굴 벡터는 부동 소수점의 배열로 저장되고 저장 시 암호화됩니다.

Amazon Rekognition Video

동영상

비디오를 분석하기 위해 Amazon Rekognition은 처리할 비디오를 서비스에 복사합니다. 비디오는 저장되어 서비스를 개선하는 데 사용될 수 있습니다. 단, AI 서비스 옵트아웃 정책 페이지를 방문하여 거기에 설명된 프로세스에 따라 옵트아웃한 경우는 예외입니다. 비디오는 AWS Key Management Service(SSE-KMS)를 사용하여 저장 시 암호화(Amazon S3)됩니다.

Amazon Rekognition Custom Labels

Amazon Rekognition Custom Labels는 저장 데이터를 암호화합니다.

이미지

모델을 학습시키기 위해 Amazon Rekognition Custom Labels는 소스 훈련 및 테스트 이미지의 사본을 만듭니다. 복사된 이미지는 사용자가 제공한 AWS KMS key나 AWS가 소유한 KMS 키를 사용한 서버 측 암호화 방식으로 Amazon Simple Storage Service(S3)에서 저장 시 암호화됩니다. Amazon Rekognition Custom Labels는 대칭 KMS 키만 지원합니다. 소스 이미지는 영향을 받지 않습니다. 자세한 내용은 Amazon Rekognition Custom Labels 모델 훈련을 참조하세요.

모델

기본적으로 Amazon Rekognition Custom Labels는 AWS 소유 키를 사용한 서버 측 암호화로 Amazon S3 버킷에 저장된 학습된 모델 및 매니페스트 파일을 암호화합니다. 자세한 내용은 서버 측 암호화를

사용하여 데이터 보호를 참조하십시오. 학습 결과는 CreateProjectVersion의 `OutputConfig` 입력 파라미터에 지정된 버킷에 기록됩니다. 학습 결과는 버킷에 대해 구성된 암호화 설정(`OutputConfig`)을 사용하여 암호화됩니다.

콘솔 버킷

Amazon Rekognition Custom Labels 콘솔은 프로젝트를 관리하는 데 사용할 수 있는 Amazon S3 버킷(콘솔 버킷)을 생성합니다. 콘솔 버킷은 기본 Amazon S3 암호화를 사용하여 암호화됩니다. 자세한 내용을 알아보려면 S3 버킷에 대한 Amazon Simple Storage Service 기본 암호화를 참조하세요. 자체 KMS 키를 사용하는 경우 콘솔 버킷을 생성한 후에 구성하십시오. 자세한 내용은 서버 측 암호화를 사용하여 데이터 보호를 참조하십시오. Amazon Rekognition Custom Labels는 콘솔 버킷에 대한 퍼블릭 액세스를 차단합니다.

Rekognition Face Liveness

Rekognition Face Liveness 서비스 계정에 저장된 모든 세션 관련 데이터는 저장 중 완전히 암호화됩니다. 기본적으로 참조 및 감사 이미지는 서비스 계정의 AWS 소유 키를 사용하여 암호화됩니다. 하지만 이러한 이미지를 암호화하기 위해 자체 AWS KMS 키를 제공하실 수도 있습니다.

전송 중 암호화

Amazon Rekognition API 엔드포인트는 HTTPS를 통한 보안 연결만을 지원합니다. 모든 통신은 TLS(전송 계층 보안)를 통해 암호화됩니다.

키 관리

AWS Key Management Service(KMS)를 사용하여 Amazon S3 버킷에 저장하는 입력 이미지 및 비디오의 키를 관리할 수 있습니다. 자세한 내용은 AWS Key Management Service 개념을 참조하세요.

Face Liveness를 위한 고객 관리형 키 암호화

CreateFaceLivenessSession API는 선택적 `KmsKeyId` 파라미터를 수용합니다. 계정에 생성한 KMS 키의 `id`를 제공할 수 있습니다. 이 키는 StartFaceLivenessSession API 중에 가져온 참조 및 감사 이미지를 암호화하는 데 사용되며 GetFaceLivenessSessionResults API 중에는 결과를 반환하기 전에 이 키를 사용하여 이미지를 복호화합니다. CreateFaceLivenessSession 요청에 OutputConfig이 포함된 경우 참조 및 감사 이미지는 지정된 Amazon S3 경로에 업로드됩니다. Amazon S3 버킷에서 서버 측 암호화(SSE-S3)를 활성화하여 저장된 데이터가 계속 암호화된 상태로 유지되도록 하는 것이 좋습니다.

자체 AWS KMS 키 ID를 입력하면 Rekognition Face Liveness 서비스는 API를 간접 호출하는 보안 주체를 대신하여 고객 관리형 키를 사용할 수 있는 권한을 얻습니다. 고객 백엔드

(CreateFaceLivenessSession 및 GetFaceLivenessSessionResults API)에서 API를 간접 호출하는 데 사용되는 보안 주체(사용자 또는 역할)는 다음을 수행할 수 있는 액세스 권한이 있어야 합니다.

- kms:DescribeKey
- kms:GenerateDataKey
- kms:Decrypt

인터네트워크 트래픽 개인 정보

Amazon Rekognition용 Amazon Virtual Private Cloud(VPC) 엔드포인트는 VPC 내의 논리적 개체로서, Amazon Rekognition에만 연결을 허용합니다. Amazon VPC는 Amazon Rekognition으로 요청을 라우팅하고, 응답을 다시 VPC로 라우팅합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 참조하세요. Amazon Rekognition과 함께 Amazon VPC 엔드포인트를 사용하는 방법에 대한 자세한 내용은 Amazon Rekognition에서 Amazon VPC 엔드포인트 사용 섹션을 참조하세요.

Amazon Rekognition에서 Amazon VPC 엔드포인트 사용

Amazon Virtual Private Cloud(VPC)를 사용하여 AWS 리소스를 호스트하는 경우, VPC와 Amazon Rekognition 간에 프라이빗 연결을 설정할 수 있습니다. 이 연결을 사용하면 Amazon Rekognition이 퍼블릭 인터넷을 통하지 않고 VPC의 리소스와 통신하게 할 수 있습니다.

Amazon VPC란 사용자가 정의한 가상 네트워크에서 AWS 리소스를 시작할 때 사용할 수 있는 AWS 서비스입니다. VPC가 있으면 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 등 네트워크 설정을 제어할 수 있습니다. VPC 엔드포인트를 통해 AWS 네트워크는 VPC와 AWS 서비스 간의 라우팅을 처리합니다.

VPC를 Amazon Rekognition에 연결하려면 Amazon Rekognition에 대해 인터페이스 VPC 엔드포인트를 정의하세요. 인터페이스 엔드포인트는 지원되는 AWS 서비스로 향하는 트래픽의 진입점 역할을 하는 프라이빗 IP 주소가 있는 탄력적 네트워크 인터페이스입니다. 이 엔드포인트를 이용하면 인터넷 게이트웨이, Network Address Translation(NAT) 인스턴스, 또는 VPN 연결 없이도 Amazon Rekognition에 안정적이고 확장 가능하게 연결됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC란 무엇입니까를 참조하세요.

인터페이스 VPC 엔드포인트는 AWS PrivateLink에 의해 활성화됩니다. 이 AWS 기술은 프라이빗 IP 주소와 함께 탄력적 네트워크 인터페이스를 사용하여 AWS 서비스 간 프라이빗 통신을 가능하게 합니다.

> **Note**
>
> 모든 Amazon Rekognition 연방 정보 처리 표준(FIPS) 엔드포인트가 AWS PrivateLink에서 지원됩니다.

Amazon Rekognition용 Amazon VPC 엔드포인트 생성

두 가지 유형의 Amazon VPC 엔드포인트를 생성하여 Amazon Rekognition에서 사용할 수 있습니다.

- Amazon Rekognition 작업에서 사용할 VPC 엔드포인트. 이것은 대부분의 사용자에게 가장 적합한 유형의 VPC 엔드포인트입니다.
- 미국 연방 정보 처리 표준(FIPS) Publication 140-2 US 정부 표준을 준수하는 엔드포인트를 사용하는 Amazon Rekognition 작업을 위한 VPC 엔드포인트

VPC에서 Amazon Rekognition 사용을 시작하려면 Amazon VPC 콘솔을 사용하여 Amazon Rekognition을 위한 인터페이스 VPC 엔드포인트를 생성합니다. 지침은 [Creating an Interface Endpoint]에서 "콘솔을 사용하여 AWS 서비스에 대한 인터페이스 엔드포인트를 생성하려면" 절차를 참조하십시오. 다음 절차 단계를 참고하십시오.

- 3단계 - 서비스 범주에서 AWS 서비스를 선택합니다.
- 4단계 - 서비스 이름에서는 다음 옵션 중 하나를 선택합니다.
  - com.amazonaws.region.rekognition - Amazon Rekognition 작업을 위한 VPC 엔드포인트를 생성합니다.
  - com.amazonaws.region.rekognition-fips - 미국 연방 정보 처리 표준(FIPS) Publication 140-2 US 정부 표준을 준수하는 엔드포인트를 사용하는 Amazon Rekognition 작업을 위한 VPC 엔드포인트를 생성합니다.

자세한 내용은 Amazon VPC 사용 설명서의 시작하기를 참조하세요.

Amazon Rekognition에 대한 VPC 엔드포인트 정책 생성

Amazon Rekognition에 대한 Amazon VPC 엔드포인트 정책을 생성하여 다음을 지정할 수 있습니다.

- 작업을 수행할 수 있는 위탁자.
- 수행할 수 있는 작업.

- 작업을 수행할 수 있는 리소스.

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

다음 예제 정책은 VPC 엔드포인트를 통해 Amazon Rekognition에 연결하는 사용자가 `DetectFaces` API 작업을 직접 호출할 수 있도록 합니다. 이 정책은 사용자가 VPC 엔드포인트를 통해 다른 Amazon Rekognition API 작업을 수행하지 못하도록 합니다.

사용자는 여전히 VPC 외부에서 다른 Amazon Rekognition API 작업을 직접 호출할 수 있습니다. VPC 외부에 있는 Amazon Rekognition API 작업에 대한 액세스를 거부하는 방법에 대한 자세한 내용은 Amazon Rekognition 자격 증명 기반 정책 섹션을 참조하세요.

JSON

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

Amazon Rekognition에 대한 VPC 엔드포인트 정책을 수정하려면

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. Amazon Rekognition에 대한 엔드포인트를 아직 생성하지 않았다면 엔드포인트 생성을 선택합니다. 그런 다음 com.amazonaws.***Region***.rekognition을 선택하고 엔드포인트 생성을 선택합니다.
3. 탐색 창에서 엔드포인트를 선택합니다.
4. com.amazonaws.***Region***.rekognition 엔드포인트를 선택하고 화면 하단의 정책 탭을 선택합니다.

5. 정책 편집을 선택하고 정책을 변경합니다.

Amazon Rekognition의 규정 준수 확인

외부 감사자는 여러 AWS 규정 준수 프로그램의 일환으로 Amazon Rekognition의 보안 및 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다.

특정 규정 준수 프로그램 범위에 속하는 AWS 서비스의 목록은 규정 준수 프로그램 제공 범위 내 AWS 서비스 단원을 참조하세요. 일반 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact를 사용하여 외부 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact 트의 보고서 다운로드를 참조하세요.

Amazon Rekognition 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서 – 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- HIPAA 보안 및 규정 준수 아키텍팅 백서 - 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 생성하는 방법을 설명합니다.
- AWS 규정 준수 리소스 – 사용자의 업계와 위치에 해당할 수 있는 워크북 및 안내서 모음입니다.
- AWS Config – 이 AWS 서비스로 리소스 구성이 내부 관행, 업계 지침 및 규정을 준수하는 정도를 평가할 수 있습니다.
- AWS Security Hub CSPM - 이 AWS 서비스는 보안 산업 표준 및 모범 사례 규정 준수 여부를 확인하는 데 도움이 되도록 AWS 내 보안 상태를 종합적으로 보여줍니다.

Amazon Rekognition의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon Rekognition은 데이터 복원성과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

Amazon Rekognition의 구성 및 취약성 분석

구성 및 IT 제어는 AWS와 고객 간의 공동 책임입니다. 자세한 내용은 AWS 공동 책임 모델을 참조하세요.

교차 서비스 혼동된 대리인 방지

에서 AWS교차 서비스 가장은 한 서비스(호출 서비스)가 다른 서비스(호출 서비스)를 호출할 때 발생할 수 있습니다. 호출 서비스는 다른 고객의 리소스에 대해 적절한 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있으며, 이로 인해 대리인 문제가 발생할 수 있습니다.

이를 방지하기 위해는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 AWS 제공합니다.

Amazon Rekognition이 리소스에 다른 서비스를 제공하는 권한을 제한하려면 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용하는 것이 좋습니다.

만약 `aws:SourceArn` 값에 Amazon S3 버킷 ARN과 같은 계정 ID가 포함되어 있지 않은 경우, 권한을 제한하려면 두 가지 키를 모두 사용해야 합니다. 두 가지 키를 모두 사용하고 `aws:SourceArn` 값이 계정 ID를 포함하는 경우, `aws:SourceAccount` 값 및 `aws:SourceArn` 값의 계정은 동일한 정책 설명문에서 사용할 경우 반드시 동일한 계정 ID를 사용해야 합니다.

하나의 리소스만 교차 서비스 액세스와 연결되도록 허용하려는 경우 `aws:SourceArn`을 사용하세요. 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 `aws:SourceAccount`를 사용하세요.

`aws:SourceArn`의 값은 Rekognition에서 사용하는 리소스의 ARN이어야 하며, `arn:aws:rekognition:region:account:resource`와 같은 형식으로 지정됩니다.

`arn:User` ARN의 값은 비디오 분석 작업을 직접 호출할 사용자(역할을 맡은 사용자)의 ARN이어야 합니다.

혼동된 대리자 문제에 대한 권장 접근 방식은 전체 리소스 ARN이 포함된 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용하는 것입니다.

리소스의 전체 ARN을 모르거나 여러 리소스를 지정하는 경우, ARN의 알 수 없는 부분에 대해 와일드카드 문자(*)를 포함한 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용합니다. 예를 들어 `arn:aws:rekognition:*:111122223333:*`입니다.

혼동된 대리인 문제를 방지하려면 다음 단계를 수행하세요.

1. IAM 콘솔의 탐색 창에서 Roles 옵션을 선택합니다. 콘솔에 현재 계정의 역할이 표시됩니다.
2. 수정할 역할의 이름을 선택합니다. 수정할 역할은 AmazonRekognitionServiceRole 권한 정책을 적용받아야 합니다. 신뢰 관계 탭을 선택합니다.
3. 신뢰 정책 편집을 선택합니다.
4. 신뢰 정책 편집 페이지에서 기본 JSON 정책을 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키 중 하나 또는 둘 다를 사용하는 정책으로 바꿉니다. 다음 정책 예시를 참조하세요.
5. 정책 업데이트를 선택합니다.

다음 예에서는 Amazon Rekognition에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용하여 혼동된 대리인 문제를 방지하는 방법을 보여줍니다.

저장된 동영상 및 스트리밍 동영상을 작업하는 경우 IAM 역할에서 다음과 같은 정책을 사용할 수 있습니다.

저장된 비디오만 다루는 경우 IAM 역할에서 다음과 같은 정책을 사용할 수 있습니다(단, `streamprocessor`를 지정하는 `StringLike` 인수를 포함시킬 필요는 없습니다).

Amazon Rekognition의 인프라 보안

관리형 서비스인 Amazon Rekognition은 AWS글로벌 네트워크 보안으로 보호됩니다. AWS보안 서비스 및가 인프라를 AWS보호하는 방법에 대한 자세한 내용은 AWS클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 환경을 설계하려면 보안 원칙 AWSWell-Architected Framework의 인프라 보호를 참조하세요AWS.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon Rekognition에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- Transport Layer Security(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

Amazon Rekognition 모니터링

모니터링은 Amazon Rekognition 및 기타 AWS 솔루션의 신뢰성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. AWS는 Rekognition을 관찰하고, 문제 발생 시 보고하고, 적절한 경우 자동 조치를 취하는 다음과 같은 모니터링 도구를 제공합니다.

- Amazon CloudWatch는 AWS 리소스와 실행 중인 애플리케이션을 AWS 실시간으로 모니터링합니다. 지표를 수집 및 추적하고, 사용자 지정 대시보드를 생성할 수 있으며, 지정된 지표가 지정한 임곗값에 도달하면 사용자에게 알리거나 조치를 취하도록 경보를 설정할 수 있습니다. 예를 들어 CloudWatch에서 Amazon EC2 인스턴스의 CPU 사용량 또는 기타 지표를 추적하고 필요할 때 자동으로 새 인스턴스를 시작할 수 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.
- Amazon CloudWatch Logs로 Amazon EC2 인스턴스, CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임계값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구성이 뛰어난 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용자 안내서를 참조하세요.
- Amazon EventBridge를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제 또는 리소스 변경과 같은 시스템 이벤트에 자동으로 대응할 수 있습니다. AWS 서비스의 이벤트는 거의 실시간으로 EventBridge로 전달됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 작업을 지정할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.
- AWS CloudTrail는 AWS 계정에 의해 또는 계정을 대신하여 수행된 API 호출 및 관련 이벤트를 캡처하고 사용자가 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 호출한 사용자 및 계정 AWS, 호출이 수행된 소스 IP 주소, 호출이 발생한 시기를 식별할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

Amazon CloudWatch를 사용한 Rekognition 모니터링

CloudWatch를 사용하면 개별 Rekognition 작업에 대한 지표 또는 계정에 대한 전역 Rekognition 지표를 가져올 수 있습니다. 지표를 사용하여 Rekognition 기반 솔루션의 상태를 추적하고 하나 이상의 지표가 정의된 임계값을 벗어날 때 통보하도록 경보를 설정할 수 있습니다. 예를 들어 발생한 서버 오류 수에 대한 측정치나 감지된 얼굴 수에 대한 측정치를 볼 수 있습니다. 또한 특정 Rekognition 작업이 성공한 횟수에 대한 지표도 볼 수 있습니다. 지표를 보려면 Amazon CloudWatch, Amazon AWS Command Line Interface 또는 CloudWatch API를 사용할 수 있습니다.

또한 Rekognition 콘솔을 사용하여 선택한 기간 동안 집계된 지표를 볼 수 있습니다. 자세한 내용은 연습 4: 집계 지표 보기(콘솔) 단원을 참조하십시오.

Rekognition에 CloudWatch 지표 사용

측정치를 사용하려면 다음 정보를 지정해야 합니다.

- 측정치 차원 또는 차원 없음. 차원은 지표를 고유하게 식별하는 데 도움이 되는 이름-값 페어입니다. Rekognition에는 작업이라는 하나의 차원이 있습니다. 이는 특정 작업에 대한 측정치를 제공합니다. 차원을 지정하지 않으면 지표의 범위가 계정 내의 모든 Rekognition 작업으로 지정됩니다.
- `UserErrorCount`와 같은 지표 이름.

AWS Management Console, AWS CLI또는 CloudWatch API를 사용하여 Rekognition의 모니터링 데이터를 가져올 수 있습니다. 또한 Amazon AWS 소프트웨어 개발 키트(SDK) 또는 CloudWatch API 도구 중 하나를 통해 CloudWatch API를 사용할 수 있습니다. 콘솔에는 CloudWatch API의 원시 데이터를 기초로 하는 일련의 그래프가 표시됩니다. 필요에 따라 콘솔에 표시되거나 API에서 가져온 그래프를 사용하는 것이 더 나을 수 있습니다.

다음 목록은 몇 가지 일반적인 지표 사용 사례를 보여 줍니다. 모든 사용 사례를 망라한 것은 아니지만 시작하는 데 참고가 될 것입니다.

방법	관련 지표
인식되는 얼굴 수를 추적하려면 어떻게 해야 합니까?	`DetectedFaceCount` 측정치의 `Sum` 통계를 모니터링합니다.
내 애플리케이션이 초당 최대 요청 수에 도달했는지 여부를 어떻게 알 수 있습니까?	`ThrottledCount` 측정치의 `Sum` 통계를 모니터링합니다.
요청 오류는 어떻게 모니터링할 수 있습니까?	`UserErrorCount` 측정치의 `Sum` 통계를 사용합니다.
총 요청 수를 찾으려면 어떻게 해야 합니까?	`ResponseTime` 측정치의 `ResponseTime` 및 `Data Samples` 통계를 사용합니다. 여기에는 오류를 초래하는 모든 요청이 포함됩니다. 성공한 작업 호출만 보려면 `SuccessfulRequestCount` 측정치를 사용합니다.

방법	관련 지표
Rekognition 작업 호출의 지연 시간은 어떻게 모니터링할 수 있습니까?	ResponseTime 측정치를 사용합니다.
IndexFaces 가 Rekognition 컬렉션에 성공적으로 얼굴을 추가한 횟수를 모니터링하려면 어떻게 해야 하나요?	SuccessfulRequestCount 측정치 및 IndexFaces 작업을 사용하여 Sum 통계를 모니터링합니다. Operation 차원을 사용하여 작업과 측정치를 선택합니다.

CloudWatch를 사용하여 Rekognition을 모니터링하려면 적절한 CloudWatch 권한이 있어야 합니다. 자세한 내용은 Amazon CloudWatch에 대한 인증 및 액세스 제어를 참조하세요.

Rekognition 지표 액세스

다음 예제에서는 CloudWatch 콘솔, AWS CLI및 CloudWatch API를 사용하여 Rekognition 지표에 액세스하는 방법을 보여줍니다.

지표를 보려면(콘솔)

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. [Metrics]를 선택하고 [All Metrics] 탭을 선택한 후 [Rekognition]을 선택합니다.
3. [Metrics with no dimensions]를 선택한 후 측정치를 선택합니다.

   예를 들어 얼마나 많은 얼굴이 감지되었는지 측정하려면 [DetectedFace] 측정치를 선택합니다.
4. 날짜 범위 값을 선택합니다. 측정치 개수는 그래프에 표시됩니다.

일정 기간 동안 성공적으로 이루어진 **DetectFaces** 작업 호출 측정치를 보려면(CLI)

- 를 열고 다음 명령을 AWS CLI 입력합니다.

```
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
```

이 예제는 일정 기간 동안 성공적으로 이루어진 `DetectFaces` 작업 호출을 보여 줍니다. 자세한 내용은 get-metric-statistics를 참조하십시오.

지표에 액세스하려면(CloudWatch API)

- `GetMetricStatistics`을 호출합니다. 자세한 내용은 CloudWatch API 참조를 참조하세요.

경보 만들기

경보 때문에 상태가 변경되면 Amazon Simple Notification Service(SNS) 메시지를 보내는 CloudWatch 경보를 생성할 수 있습니다. 경보는 지정한 기간에 단일 지표를 감시하고 여러 기간에 지정된 임곗값에 대한 지표 값을 기준으로 작업을 하나 이상 수행합니다. 이 작업은 Amazon SNS 주제 또는 Auto Scaling 정책에 전송되는 알림입니다.

경보는 지속적인 상태 변경에 대해서만 작업을 호출합니다. CloudWatch 경보는 단순히 특정 상태에 있다고 해서 작업을 호출하지 않습니다. 상태가 변경되어 지정된 기간 수 동안 유지되어야 합니다.

경보를 설정하려면(콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/ CloudWatch 콘솔을 엽니다.
2. 경보 생성을 선택합니다. 그러면 Create Alarm Wizard가 시작됩니다.
3. [Metrics with no dimensions] 측정치 목록에서 [Rekognition Metrics]를 선택한 후 측정치를 선택합니다.

   예를 들어 감지된 얼굴의 최대 수에 대한 경보를 설정하려면 [DetectedFaceCount]를 선택합니다.
4. [Time Range] 영역에서 호출한 얼굴 감지 작업이 포함된 날짜 범위 값을 선택합니다. 다음을 선택합니다.
5. [Name]과 [Description]을 입력합니다. [Whenever]에서 [>=]를 선택하고 원하는 최대값을 입력합니다.
6. 경보 상태에 도달할 때 CloudWatch에서 이메일을 보내도록 하려면 이 경보가 발생할 경우 항상:에서 상태가 ALARM입니다를 선택합니다. 기존 Amazon SNS 주제에 경보를 전송하려면 다음 주소로 알림 전송:에서 기존 SNS 주제를 선택합니다. 새 이메일 구독 목록에 이름과 이메일 주소를 설정하려면 주제 생성을 선택합니다. 나중에 경보를 설정하는 데 사용할 수 있도록 CloudWatch가 목록을 저장하고 필드에 표시합니다.

> **Note**
>
> 새 Amazon SNS 주제를 생성하기 위해 주제 생성을 사용할 경우 의도한 수신자가 알림을 받기 전에 이메일 주소가 확인되어야 합니다. Amazon SNS는 경보가 경보 상태에 진입할 때만 이메일을 전송합니다. 이러한 경보 상태 변경이 이메일 주소 확인 전에 발생할 경우, 의도된 수신자는 알림을 받지 못합니다.

7. 알람 미리보기 섹션에서 경보를 미리 볼 수 있습니다. 경보 생성을 선택합니다.

경보를 설정하려면(AWS CLI)

- 를 열고 다음 명령을 AWS CLI 입력합니다. `alarm-actions` 파라미터의 값을 변경하여 이전에 만든 Amazon SNS 주제를 참조하세요.

  ```
  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
  ```

  이 예제는 5분 이내에 10회 이상 사용자 오류가 발생하는 경우의 경보를 생성하는 방법을 보여 줍니다. 자세한 내용은 put-metric-alarm을 참조하십시오.

경보를 설정하려면(CloudWatch API)

- `PutMetricAlarm`를 호출합니다. 자세한 내용은 CloudWatch API 참조를 참조하세요.

Rekognition의 CloudWatch 지표

이 섹션에는 Amazon Rekognition에 사용할 수 있는 Amazon CloudWatch 지표 및 작업 차원에 대한 정보가 나와 있습니다.

Rekognition 콘솔에서 Rekognition 지표의 집계 보기도 확인할 수 있습니다. 자세한 내용은 연습 4: 집계 지표 보기(콘솔) 단원을 참조하십시오.

Rekognition의 CloudWatch 지표

다음 표에는 Rekognition 지표가 요약되어 있습니다.

지표	설명
SuccessfulRequestCount	성공한 요청 수. 성공적 요청의 응답 코드 범위는 200 - 299입니다. 단위: 수 유효한 통계: `Sum,Average`
ThrottledCount	제한된 요청 수. Rekognition은 계정에 대해 설정된 초당 트랜잭션 한도를 초과하는 요청이 수신되면 요청을 제한합니다. 계정에 대해 설정된 한도가 자주 초과되면 한도 증가를 요청할 수 있습니다. 증가를 요청하려면 AWS 서비스 한도를 참조하십시오. 단위: 수 유효한 통계: `Sum,Average`
ResponseTime	Rekognition이 응답을 계산하는 시간(밀리초) 단위: 1. `Data Samples` 통계 개수 2. `Average` 통계의 밀리초 유효한 통계: `Data Samples,Average` **Note** `ResponseTime` 지표는 Rekognition 지표 창에 포함되지 않습니다.
DetectedFaceCount	`IndexFaces` 또는 `DetectFaces` 작업으로 감지된 얼굴의 수. 단위: 수

지표	설명
	유효한 통계: `Sum,Average`
DetectedLabelCount	`DetectLabels` 작업을 사용하여 감지된 레이블 수. 단위: 수 유효한 통계: `Sum,Average`
ServerErrorCount	서버 오류 수. 서버 오류의 응답 코드 범위는 500 - 599입니다. 단위: 수 유효한 통계: `Sum, Average`
UserErrorCount	사용자 오류 수(잘못된 파라미터, 잘못된 이미지, 권한 없음 등). 사용자 오류의 응답 코드 범위는 400~499입니다. 단위: 수 유효한 통계: `Sum,Average`
MinInferenceUnits	`StartProjectVersion` 요청 중에 지정된 추론 단위의 최소 개수 단위: 수 유효한 통계: `Average`
MaxInferenceUnits	`StartProjectVersion` 요청 중에 지정된 추론 단위의 최대 개수 단위: 수 유효한 통계: `Average`
DesiredInferenceUnits	Rekogniti이 스케일 업 또는 다운하는 추론 단위의 수 단위: 수 유효한 통계: `Average`

지표	설명
InServiceInference Units	모델이 사용하는 추론 단위의 수 단위: 수 유효한 통계: `Average` Average 통계를 사용하여 사용된 인스턴스 수에 대한 1분 평균을 구하는 것을 권장합니다.

Rekognition Streaming의 CloudWatch 지표

Rekognition에는 스트리밍 작업에 사용되는 두 번째 네임스페이스인 “Rekognition Streaming”도 있습니다. 다음 표에는 Rekognition Streaming 지표가 요약되어 있습니다.

지표	설명
SuccessfulRequestCount	성공한 요청 수. 성공적 요청의 응답 코드 범위는 200 - 299입니다. 단위: 수 유효한 통계: `Sum,Average`
CallCount	계정에서 수행된 지정된 작업 수입니다. 유효한 통계: `Sum,Average`
ThrottledCount	제한된 요청 수. Rekognition은 계정에 대해 설정된 초당 트랜잭션 한도를 초과하는 요청이 수신되면 요청을 제한합니다. 계정에 대해 설정된 한도가 자주 초과되면 한도 증가를 요청할 수 있습니다. 증가를 요청하려면 AWS 서비스 한도를 참조하십시오. 단위: 수 유효한 통계: `Sum,Average`
ServerErrorCount	서버 오류 수. 서버 오류의 응답 코드 범위는 500 - 599입니다. 단위: 수

지표	설명
	유효한 통계: `Sum, Average`
UserErrorCount	사용자 오류 수(잘못된 파라미터, 잘못된 이미지, 권한 없음 등). 사용자 오류의 응답 코드 범위는 400~499입니다. 단위: 수 유효한 통계: `Sum,Average`
ConcurrentSessions	계정이 현재 사용 중인 동시 세션 수입니다. 단위: 수 유효한 통계: `Max,Average`

Rekognition에 대한 CloudWatch 차원

작업별 측정치를 검색하려면 `Rekognition` 네임스페이스를 사용하고 operation 차원을 제공합니다.

차원에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서에서 차원을 참조하세요.

Rekognition Custom Labels에 대한 CloudWatch 차원

다음 표에는 Rekognition Custom Labels와 함께 사용할 수 있는 CloudWatch 차원이 나와 있습니다.

차원	설명
ProjectName	`CreateProject` 로 생성한 Rekognition Custom Labels 프로젝트의 이름
VersionName	`CreateProjectVersion` 으로 생성한 Rekognition Custom Labels 프로젝트 버전의 이름

차원에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서에서 차원을 참조하세요.

를 사용하여 Amazon Rekognition API 호출 로깅 AWS CloudTrail

Amazon Rekognition은 Amazon Rekognition에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 Amazon Rekognition에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. 캡처된 직접 호출에는 Amazon Rekognition 콘솔로부터의 직접 호출과 Amazon Rekognition API 작업에 대한 코드 직접 호출이 포함됩니다. 추적을 생성하면 Amazon Rekognition 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 전송할 수 있습니다. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 Amazon Rekognition에 보낸 요청, 요청을 보낸 IP 주소, 요청한 사람, 요청한 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail의 Amazon Rekognition 정보

AWS 계정을 생성할 때 계정에서 CloudTrail이 활성화됩니다. Amazon Rekognition에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다. 자세한 설명은 CloudTrail 이벤트 기록으로 이벤트 보기를 참조하세요.

Amazon Rekognition에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 리전에 추적이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에서 Amazon SNS 알림 구성
- 여러 리전으로부터 CloudTrail 로그 파일 받기 및 여러 계정으로부터 CloudTrail 로그 파일 받기

모든 Amazon Rekognition 작업은 CloudTrail에서 로깅되며 Amazon Rekognition API 참조에 기록됩니다. 예를 들어 `CreateCollection`, `CreateStreamProcessor`, `DetectCustomLabels` 작업을 직접 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 관한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 설명은 CloudTrail userIdentity 요소를 참조하세요.

Amazon Rekognition 로그 파일 항목 이해

트레일이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음은 `StartLabelDetection` 및 `DetectLabels` API 작업의 CloudTrail 로그 항목을 보여주는 예제입니다.

```
{
    "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 Rekognition의 가이드라인 및 할당량

다음 섹션에는 Amazon Rekognition 사용과 관련된 가이드라인과 할당량이 나와 있습니다. 할당량에는 2가지 종류가 있습니다. 최대 이미지 크기와 같은 고정된 할당량은 변경할 수 없습니다. AWS Service Quotas 페이지에 나열된 기본 할당량은 기본 할당량 섹션에 설명된 절차에 따라 변경할 수 있습니다.

주제

- 지원되는 리전
- 고정된 할당량
- 기본 할당량

지원되는 리전

Amazon Rekognition을 사용할 수 있는 모든 AWS 리전 목록은 Amazon Web Services 일반 참조의 AWS 리전 및 엔드포인트를 참조하세요.

고정된 할당량

Amazon Rekognition 제한 사항 목록은 다음과 같습니다. TPS(초당 트랜잭션) 제한과 같이 변경이 가능한 제한 사항에 대한 자세한 내용은 기본 할당량 섹션을 참조하세요.

Amazon Rekognition Custom Labels 제한 사항은 Amazon Rekognition Custom Labels 가이드라인 및 할당량을 참조하세요.

Amazon Rekognition Image

- Amazon S3 객체로 저장되는 이미지 최대 크기는 15MB로 제한됩니다.
- `DetectModerationLabels`의 최대 이미지 크기는 너비와 높이 모두 10,000픽셀입니다.
- `DetectLabels`의 최대 이미지 크기는 너비와 높이 모두 10,000픽셀입니다.
- 감지하려면 얼굴이 1920x1080픽셀의 이미지에서 40x40픽셀보다 작아서는 안 됩니다. 1920X1080 픽셀보다 높은 차원의 이미지는 비례적으로 더 큰 최소 얼굴 크기가 필요합니다.
- 최소 이미지 크기는 높이 및 너비 모두 80픽셀입니다. `DetectProtectiveEquipment`의 최소 이미지 크기는 높이 및 너비 모두 64픽셀입니다.
- `DetectProtectiveEquipment`의 최대 이미지 크기는 너비와 높이 모두 4,096픽셀입니다.

- `DetectProtectiveEquipment`로 감지하려면 800x1,300픽셀의 이미지에서 사람이 100x100픽셀보다 작아서는 안 됩니다. 800X1,300픽셀보다 크기가 큰 이미지는 그에 비례해 최소 인물 크기가 더 커져야 합니다.
- 파라미터로 API에 전달되는 원시 바이트의 최대 이미지 크기는 5MB입니다. `DetectProtectiveEquipment` API의 한도는 4MB입니다.
- Amazon Rekognition은 PNG 및 JPEG 이미지 형식을 지원합니다. 즉, `DetectLabels` 및 `IndexFaces` 등 다양한 API 작업에 입력으로 제공하는 이미지는 지원되는 다음 형식 중 하나여야 합니다.
- 단일 얼굴 컬렉션에 저장할 수 있는 최대 얼굴 벡터의 수는 2천만 개입니다.
- 단일 얼굴 컬렉션에 저장할 수 있는 최대 사용자 벡터의 수는 1천만 개입니다.
- 검색 API가 반환하는 일치 얼굴 벡터의 최대 수는 4,096개입니다.
- 검색 API가 반환하는 일치 사용자 벡터의 최대 수는 4,096개입니다.
- `DetectText`는 이미지에서 최대 100개 단어를 감지할 수 있습니다.
- `DetectProtectiveEquipment`는 최대 15명의 PPE(개인 보호 장비)를 감지할 수 있습니다.

이미지 및 얼굴 비교에 대한 모범 사례 정보는 센서, 입력 이미지 및 비디오 모범 사례 단원을 참조하십시오.

Amazon Rekognition Image 대량 분석

- Amazon Rekognition Image 대량 분석은 최대 10k 이미지 크기의 이미지 배치를 분석할 수 있습니다.
- Amazon Rekognition Image 대량 분석은 최대 50MB 크기의 입력 매니페스트를 지원합니다.

Amazon Rekognition Video 저장 동영상

- Amazon Rekognition Video는 저장된 비디오를 최대 10GB 크기까지 분석할 수 있습니다.
- Amazon Rekognition Video는 저장된 비디오를 최대 6시간 길이까지 분석할 수 있습니다.
- Amazon Rekognition Video는 계정당 최대 20개의 동시 작업을 지원합니다.
- 저장된 비디오는 H.264 코덱을 사용하여 인코딩해야 합니다. 지원되는 파일 형식은 MPEG-4와 MOV입니다.
- 오디오 데이터를 분석하는 모든 Amazon Rekognition Video API는 AAC 오디오 코덱만 지원합니다.

- 매김 토큰의 TTL(Time To Live) 기간은 24시간입니다. 매김 토큰은 `NextToken` 등 Get 작업으로 반환된 `GetLabeldetection` 필드에 있습니다.

Amazon Rekognition Video 스트리밍 비디오

- Kinesis Video 입력 스트림은 최대 1개의 Amazon Rekognition Video 스트림 프로세서와 연결될 수 있습니다.
- Kinesis Video 출력 스트림은 최대 1개의 Amazon Rekognition Video 스트림 프로세서와 연결될 수 있습니다.
- Amazon Rekognition Video 스트림 프로세서와 연결된 Kinesis Video 입력 스트림 및 Kinesis Data 출력 스트림은 여러 프로세서와 공유될 수 없습니다.
- 오디오 데이터를 분석하는 모든 Amazon Rekognition Video API는 ACC 오디오 코덱만 지원합니다.

기본 할당량

기본 할당량은 AWS Service Quotas에서 확인할 수 있습니다. 이 한도는 기본값이며 변경할 수 있습니다. 한도 제한을 높이도록 요청하려면 사례를 생성합니다. 현재 할당량 한도(적용된 할당량 값)를 보려면 Amazon Rekognition Service Quotas를 참조하세요. Amazon Rekognition Image API의 TPS 활용 기록을 보려면 Amazon Rekognition Service Quotas 페이지를 참조해 특정 API 작업을 선택하여 해당 작업에 대한 기록을 확인하세요.

주제

- TPS 할당량 변경 계산
- TPS 할당량 모범 사례
- TPS 할당량 변경을 위한 사례 생성

TPS 할당량 변경 계산

요청하려는 새 한도는 얼마인가요? 초당 트랜잭션(TPS)은 예상 워크로드가 최고조에 달할 때 가장 적합합니다. 워크로드가 최고조에 달했을 때의 최대 API 동시 직접 호출과 응답 시간(5~15초)을 파악하는 것이 중요합니다. 최소 5초가 되어야 한다는 점에 유의하세요. 다음은 두 가지 예입니다.

- 예 1: 가장 바쁜 시간이 시작될 때 예상되는 최대 동시 얼굴 인증(CompareFaces API) 사용자는 1,000명입니다. 이러한 응답은 10초에 걸쳐 분산됩니다. 따라서 해당 리전의 CompareFaces API에 필요한 TPS는 100(1,000/10)입니다.

- 예 2: 가장 바쁜 시간이 시작될 때 예상되는 최대 동시 객체 감지(DetectLabels API) 직접 호출은 250입니다. 이러한 응답은 5초에 걸쳐 분산됩니다. 따라서 해당 리전의 DetectLabels API에 필요한 TPS는 50(250/5)입니다.

TPS 할당량 모범 사례

초당 트랜잭션(TPS)의 권장 모범 사례에는 트래픽 급증 완화, 재시도 구성, 지수 백오프 및 지터 구성 등이 포함됩니다.

1. 트래픽 급증을 완화하세요. 급증하는 트래픽은 처리량에 영향을 줍니다. 할당된 초당 트랜잭션(TPS)에서 처리량을 극대화하려면 큐잉 서버리스 아키텍처 또는 다른 메커니즘을 사용하여 트래픽이 더 고르도록 '평탄화'하세요. Rekognition을 사용한 서버리스 대규모 이미지 및 비디오 처리에 대한 코드 샘플 및 참조는 Amazon Rekognition을 사용한 대규모 이미지 및 비디오 처리를 참조하세요.
2. 재시도를 구성합니다. the section called "오류 처리"의 가이드라인에 따라 재시도를 허용하는 오류에 대해 재시도를 구성하세요.
3. 지수 백오프 및 지터를 구성하세요. 재시도를 구성할 때 지수 백오프 및 지터를 함께 구성하면 달성 가능한 처리량을 높일 수 있습니다. AWS의 오류 재시도 횟수 및 지수 백오프를 참조하세요.

TPS 할당량 변경을 위한 사례 생성

사례를 생성하려면 사례 생성으로 이동하여 다음 질문에 답하세요.

- 트래픽 급증을 완화하고 재시도, 지수 백오프, 지터를 구성하기 위해 the section called "TPS 할당량 모범 사례"를 구현해 보셨나요?
- TPS 할당량 변경이 얼마나 필요한지 계산해 보셨나요? 그렇지 않은 경우 the section called "TPS 할당량 변경 계산" 섹션을 참조하세요.
- 향후 요구 사항을 더 정확하게 예측하기 위해 TPS 사용 기록을 확인해 보셨나요? TPS 사용 기록을 보려면 Amazon Rekognition Service Quotas 페이지를 참조하세요.
- 귀하의 사용 사례는 무엇인가요?
- 어떤 API를 사용하실 예정인가요?
- 이 API를 사용할 리전은 어디인가요?
- 여러 리전으로 로드를 분산할 수 있나요?
- 하루에 처리하는 이미지 수는 몇 개인가요?

- 이 볼륨이 얼마나 오래 지속될 것으로 예상하시나요(급증이 일회성인가요, 아니면 계속되나요)?
- 기본 한도에 따른 차단의 양상은 어떤가요? 다음 예외 테이블을 검토하여 해당하는 시나리오를 확인하세요.

오류 코드	예외	메시지	이것은 무엇을 의미하나요?	재시도 가능한가요?
HTTP 상태 코드 400	ProvisionedThroughputExceededException	프로비저닝된 속도가 초과되었습니다.	제한되었음을 나타냅니다. 재시도하거나 한도 증가 요청을 고려할 수 있습니다.	예
HTTP 상태 코드 400	ThrottlingException	Slow down; sudden increase in rate of requests.	전송하는 트래픽이 급증하여 제한되었을 수 있습니다. 트래픽을 조정하여 더 고르고 일관되게 만들어야 합니다. 그 후 추가로 재시도를 구성하세요. 모범 사례를 참조하세요.	예
HTTP 상태 코드 5xx	ThrottlingException(HTTP 500)	Service Unavailable	작업을 지원하기 위해 백엔드가 스케일 업 되고 있음을 나타냅니다. 요청을 다시 시도해야 합니다.	예

오류 코드에 대한 자세한 내용은 the section called “오류 처리” 섹션을 참조하세요.

> **Note**
>
> 이러한 한도는 사용자가 있는 리전에 따라 다릅니다. 한도 변경을 위해 사례를 만들면 요청한 리전에서 사용자가 요청하는 API 작업에 영향을 미칩니다. 다른 API 작업 및 리전은 영향을 받지 않습니다.

Amazon Rekognition의 문서 기록

다음 표는 Amazon Rekognition 개발자 안내서의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

- 문서 최종 업데이트: 2023년 6월 15일

변경 사항	설명	날짜
Amazon Rekognition은 이제 이미지의 콘텐츠 조절에 대해 새로운 조절 레이블과 향상된 정확도를 지원합니다.	Amazon Rekognition의 콘텐츠 조절 기능이 향상된 정확도, 새 레이블 감지, 애니메이션 및/또는 일러스트 콘텐츠를 식별하는 기능을 제공합니다.	2024년 2월 1일
Amazon Rekognition에서 이제 대량 이미지 분석을 지원	Amazon Rekognition은 이제 StartMediaAnalysisJob 작업으로 매니페스트 파일을 사용한 대규모 이미지 컬렉션의 비동기적 처리를 지원합니다.	2023년 10월 23일
Amazon Rekognition에서 이제 어댑터를 사용한 사용자 지정 콘텐츠 조절을 지원	Amazon Rekognition은 이제 기존 Rekognition 딥 러닝 모델의 기능을 확장하는 어댑터를 사용하여 DetectModerationLabels API의 향상된 정확도를 지원합니다.	2023년 10월 12일
Rekognition에서 이제 컬렉션에 사용자 벡터를 지원	Rekognition 얼굴 컬렉션은 이제 사용자 벡터 생성을 지원합니다. 사용자 벡터는 동일한 사용자의 여러 얼굴 벡터를 집계하므로 사용자를 보다 강력하게 파악하여 정확도를 향상시킵니다.	2023년 6월 12일

사용자 관리를 위한 작업이 AmazonRekognitionReadOnlyAccess 관리형 정책에 추가됨	Amazon Rekognition의 `AmazonRekognitionReadOnlyAccess` 관리형 정책에 `ListUsers` , `SearchUsers` , `SearchUsersByImage` 작업이 추가되었습니다.	2023년 6월 12일
Amazon Rekognition Image로 이제 시선 방향을 유추 가능	Amazon Rekognition Image의 얼굴 감지 작업이 개선되어 이제 감지된 얼굴의 시선 방향을 유추할 수 있습니다.	2023년 5월 31일
Rekognition 콘텐츠 조절 API가 개선됨	Rekognition은 이미지 및 동영상 조절을 위한 콘텐츠 조절 모델을 개선했습니다. 이 개선 사항으로 노골적, 폭력적, 선정적인 콘텐츠의 탐지 범위가 크게 확대되었습니다. 이제 고객은 노골적이고 폭력적인 콘텐츠를 더 정확하게 탐지하여 최종 사용자 경험을 개선하고 브랜드 정체성을 보호하며 모든 콘텐츠가 업계 규정 및 정책을 준수하는지 확인할 수 있습니다.	2023년 5월 9일
Amazon Rekognition Image로 이제 가려진 얼굴을 감지 가능	Amazon Rekognition Image는 이제 가려진 얼굴을 감지할 수 있습니다. Amazon Rekognition Image의 DetectFaces 및 IndexFaces API는 새로운 FaceOccluded 속성을 반환합니다. 이 속성은 물체, 옷 및 신체 부위가 겹쳐져 이미지 내 얼굴이 부분적으로만 보이거나 완전히 보이지 않는지를 나타냅니다.	2023년 5월 5일

Rekognition으로 이제 얼굴 생체 확인 감지 가능	이제 Amazon Rekognition Video를 사용하여 비디오의 생체 확인을 감지하여 카메라 앞의 사용자가 실제로 존재하는지 확인할 수 있습니다. 또한 Face Liveness 탐지기는 카메라에 가해지거나 카메라를 우회하려는 스푸핑 공격을 탐지합니다.	2023년 4월 11일
Amazon Rekognition Video 업데이트	Amazon Rekognition Video는 이제 더 많은 레이블을 탐지하고 이미지 및 레이블의 속성에 대한 더 많은 정보를 반환할 수 있습니다. GetLabelDetection API는 이제 별칭 및 카테고리에 대한 정보를 반환합니다. 반환된 레이블 정보는 포함 및 제외 필터 옵션을 사용하여 필터링할 수 있습니다. 타임스탬프 또는 비디오 세그먼트별로 결과를 집계할 수 있습니다.	2022년 12월 7일
Amazon Rekognition Image 업데이트	Amazon Rekognition Image는 이제 더 많은 레이블을 탐지할 수 있으며 이미지 및 레이블의 속성에 대한 더 많은 정보를 반환합니다. DetectLabels API는 이제 별칭, 카테고리 및 주 색상과 같은 이미지 속성에 관한 정보를 반환합니다. 반환된 레이블 정보는 포함 및 제외 필터 옵션을 사용하여 필터링할 수 있습니다.	2022년 11월 11일

ProjectPolicy 및 사용자 지정 레이블 모델 복사를 위한 작업이 AmazonRekognitionReadOnlyAccess 관리형 정책에 추가됨	Amazon Rekognition의 AmazonRekognitionReadOnlyAccess 관리형 정책에 ListProjectPolicies 작업이 추가되었습니다.	2022년 7월 21일
ProjectPolicy 및 사용자 지정 레이블 모델 복사를 위한 작업이 AmazonRekognitionFullAccess 및 AmazonRekognitionCustomLabelsFullAccess 관리형 정책에 추가됨	Rekognition에서 CopyProjectVersion , PutProjectPolicy , ListProjectPolicies , DeleteProjectPolicy 작업이 AmazonRekognitionCustomLabelsFullAccess 및 AmazonRekognitionFullAccess 관리형 정책에 추가되었습니다.	2022년 7월 21일
Amazon Rekognition Video에서 이제 스트리밍 비디오의 레이블 감지 가능	Amazon Rekognition Video는 이제 스트리밍 비디오에서 반려동물과 패키지 등의 레이블을 감지할 수 있습니다. 이는 CreateStreamProcessor 작업 시 생성된 스트림 프로세서의 ConnectedHome 설정을 사용하여 수행됩니다.	2022년 4월 28일
Amazon Rekognition 개발자 안내서에서 API 참조 제거	Amazon Rekognition API 참조는 이제 Amazon Rekognition API 참조에서 사용 가능합니다.	2022년 2월 24일

다음 관리형 정책에 대한 데이터 세트 관리 업데이트: AWS 관리형 정책: AmazonRekognitionReadOnlyAccess,AWS 관리형 정책: AmazonRekognitionFullAccess,AWS 관리형 정책: AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition에서 `CreateDataset` , `ListDatasetEntries` , `ListDatasetLabels` , `DescribeDataset` , `UpdateDatasetEntries` , `DistributeDatasetEntries` , DeleteDataset 작업이 `AmazonRekognitionReadOnlyAccess` , `AmazonRekognitionFullOnlyAccess` , `AmazonRekognitionCustomLabelsFullAccess` 관리형 정책에 추가되었습니다.	2021년 11월 1일
목차의 새 노드에서 GitHub에서 호스팅되는 Amazon Rekognition 예제 표시	이제 코드 예제 리포지토리의 업데이트된 AWS 코드 예제가 Amazon Rekognition 개발자 안내서의 별도의 노드에 표시되므로 더 쉽게 액세스할 수 있습니다.	2021년 10월 22일
Amazon Rekognition에서 비디오 세그먼트의 블랙 프레임 및 기본 프로그램 콘텐츠 감지 가능	Amazon Rekognition은 `StartSegmentDetection` 및 `GetSegmentDetection` 작업을 사용하여 블랙 프레임, 색상 막대, 오프닝 크레딧, 엔딩 크레딧, 스튜디오 로고, 기본 프로그램 콘텐츠를 비디오의 기술적 큐로 식별할 수 있습니다.	2021년 6월 7일

다음 관리형 정책에 대한 데이터 세트 관리 업데이트:	Amazon Rekognition `DetectText` 작업을 사용하여 이미지에서 최대 100개의 단어를 감지할 수 있습니다.	2021년 5월 21일
AmazonRekognitionReadOnlyAccess 및 AmazonRekognitionFullAccess에 대한 태그 지정 업데이트	Rekognition에서 `AmazonRekognitionFullAccess` 및 `AmazonRekognitionReadOnlyAccess` 정책에 새로운 태그 지정 작업을 추가했습니다.	2021년 4월 2일
Amazon Rekognition에서 이제 태그 지정을 지원	이제 태그를 사용하여 Amazon Rekognition 컬렉션, 스트림 프로세서 및 사용자 지정 레이블 모델을 식별, 구성, 검색 및 필터링할 수 있습니다.	2021년 3월 25일
Amazon Rekognition에서 이제 개인 보호 장비 탐지 가능	Amazon Rekognition은 이제 이미지에서 사람의 손 가리개, 얼굴 가리개 및 머리 덮개를 감지할 수 있습니다.	2020년 10월 15일
Amazon Rekognition에 콘텐츠 조절 카테고리 신설	Amazon Rekognition 콘텐츠 조절 카테고리에는 이제 마약, 담배, 주류, 도박, 무례한 제스처, 증오 기호 등 6개의 새로운 카테고리가 포함됩니다.	2020년 10월 12일
Kinesis Video Streams의 Amazon Rekognition Video 결과를 로컬로 표시하기 위한 새로운 자습서	Kinesis Video Streams의 스트리밍 비디오에서 출력되는 Amazon Rekognition Video 결과물을 로컬 비디오 피드에 표시할 수 있습니다.	2020년 7월 20일

Gstreamer 사용을 위한 새로운 Amazon Rekognition 자습서	Gstreamer를 사용하면 Kinesis Video Streams를 통해 디바이스 카메라 소스의 라이브스트림 비디오를 Amazon Rekognition Video로 수집할 수 있습니다.	2020년 7월 17일
Amazon Rekognition에서 이제 저장된 비디오의 세그먼트화 지원	비동기 방식 Amazon Rekognition Video 세그먼트화 API를 사용하면 저장된 비디오에서 블랙 프레임, 색상 막대, 엔딩 크레딧 및 샷을 감지할 수 있습니다.	2020년 6월 22일
Amazon Rekognition에서 이제 Amazon VPC 엔드포인트 정책 지원	정책을 지정하여 Amazon Rekognition Amazon VPC 엔드포인트에 대한 액세스를 제한할 수 있습니다.	2020년 3월 3일
Amazon Rekognition에서 이제 저장된 동영상의 텍스트 감지를 지원	Amazon Rekognition Video API를 사용하여 저장된 비디오에서 텍스트를 비동기적으로 감지할 수 있습니다.	2020년 2월 17일
Amazon Rekognition에서 이제 Augmented AI(미리 보기) 및 Amazon Rekognition Customs Labels를 지원	Amazon Rekognition Customs Labels를 사용하면 자체의 기계 학습 모델을 생성하여 이미지에서 특수 객체, 장면 및 개념을 감지할 수 있습니다. DetectModerationLabels는 이제 Amazon Augmented AI(프리뷰)를 지원합니다.	2019년 12월 3일
Amazon Rekognition에서 이제 AWS PrivateLink를 지원	이제 AWS PrivateLink를 통해 VPC와 Amazon Rekognition 간에 프라이빗 연결을 설정할 수 있습니다.	2019년 9월 12일

Amazon Rekognition 얼굴 필터링	Amazon Rekognition은 IndexFaces API 작업에 향상된 얼굴 필터링 지원을 추가하고 CompareFaces 및 SearchFacesByImage API 작업에 얼굴 필터링을 도입합니다.	2019년 9월 12일
Amazon Rekognition Video 예제 업데이트	Amazon Rekognition Video 예제 코드가 Amazon SNS 주제 및 Amazon SQS 대기열을 생성하고 구성하도록 업데이트되었습니다.	2019년 9월 5일
Ruby 및 Node.js 예제 추가	동기식 레이블 및 얼굴 감지를 위해 Amazon Rekognition Image Ruby 및 Node.js 예제가 추가되었습니다.	2019년 8월 19일
안전하지 않은 콘텐츠 감지 업데이트	Amazon Rekognition 안전하지 않은 콘텐츠 감지 기능이 이제 폭력적인 콘텐츠를 감지할 수 있습니다.	2019년 8월 9일
GetContentModeration 작업 업데이트	GetContentModeration은 이제 안전하지 않은 콘텐츠를 감지하는 데 사용되는 조절 감지 모델의 버전을 반환합니다.	2019년 2월 13일
GetLabelDetection 및 DetectModerationLabels 작업 업데이트	GetLabelDetection은 이제 일반 객체에 대한 경계 상자 정보와 감지된 레이블의 계층적 분류를 반환합니다. 이제 레이블 감지에 사용된 모델 버전이 반환됩니다. DetectModerationLabels는 안전하지 않은 콘텐츠를 감지하는 데 사용되는 모델의 버전을 반환합니다.	2019년 1월 17일

DetectFaces 및 IndexFaces 작업 업데이트	이 릴리스에서는 DetectFaces 및 IndexFaces 작업이 업데이트되었습니다. 속성 입력 파라미터를 모두로 설정할 경우, 얼굴 위치 랜드마크에 upperJawlineLeft, midJawlineLeft, chinBottom, midJawlineRight, upperJawlineRight 등 5개의 새 랜드마크가 포함됩니다.	2018년 11월 19일
DetectLabels 작업 업데이트	이제 특정 객체에 대해 경계 상자가 반환됩니다. 이제 레이블에 계층적 분류를 사용할 수 있습니다. 이제 사용되는 감지 모델의 버전을 가져올 수 있습니다.	2018년 11월 1일
IndexFaces 작업 업데이트	IndexFaces를 통해 이제 QualityFilter 입력 파라미터를 사용하여 저화질로 감지된 얼굴을 필터링할 수 있습니다. 또한 MaxFaces 입력 파라미터를 사용하여 얼굴 감지의 화질과 감지된 얼굴 크기를 기반으로 반환되는 얼굴 수를 줄일 수 있습니다.	2018년 9월 18일
DescribeCollection 작업 추가	DescribeCollection 작업을 호출하여 기존 모음에 관한 정보를 얻을 수 있습니다.	2018년 8월 22일
새 Python 예제	Python 예제가 Amazon Rekognition Video 콘텐츠에 추가되고 일부 콘텐츠가 재구성되었습니다.	2018년 6월 26일

콘텐츠 레이아웃 업데이트	Amazon Rekognition Image 콘텐츠가 재구성되고 새로운 Python 및 C# 예제가 추가되었습니다.	2018년 5월 29일
Amazon Rekognition에서 지원 AWS CloudTrail	Amazon Rekognition은 Amazon Rekognition에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. 자세한 내용은 AWS CloudTrail로 Amazon Rekognition API 호출 로깅을 참조하십시오.	2018년 4월 6일
저장된 비디오와 스트리밍 비디오 분석. 새로운 목차	저장된 비디오 분석에 대한 자세한 내용은 저장된 비디오 작업을 참조하십시오. 스트리밍 비디오 분석에 대한 자세한 내용은 스트리밍 비디오 작업을 참조하십시오. Amazon Rekognition 설명서의 목차가 이미지와 비디오 작업에 대한 내용을 반영해 재구성되었습니다.	2017년 11월 29일
이미지 및 얼굴 인식 모델의 텍스트	Amazon Rekognition은 이제 이미지에서 텍스트를 감지할 수 있습니다. 자세한 내용은 텍스트 감지를 참조하세요. Amazon Rekognition은 얼굴 감지 딥 러닝 모델의 버전 관리를 도입합니다. 자세한 내용은 모델 버전 관리를 참조하십시오.	2017년 11월 21일

유명 인사 인식	Amazon Rekognition은 이제 이미지를 분석하여 유명 인사를 인식할 수 있습니다. 자세한 내용은 유명 인사 인식을 참조하십시오.	2017년 6월 8일
이미지 조절	Amazon Rekognition은 이제 이미지에 노골적이거나 선정적인 성인 콘텐츠가 포함되어 있는지 판단할 수 있습니다. 자세한 내용은 비안전 콘텐츠 감지를 참조하십시오.	2017년 4월 19일
감지된 얼굴의 연령 범위. Rekognition 지표 집계 창	Amazon Rekognition은 이제 Rekognition API가 감지한 얼굴의 추정 연령 범위(단위: 세)를 반환합니다. 자세한 내용은 AgeRange를 참조하십시오. 이제 Rekognition 콘솔에는 지정된 기간 동안 Rekognition의 Amazon CloudWatch 지표 집계를 위한 활동 그래프를 보여주는 지표 창이 있습니다. 자세한 내용은 연습 4: 집계 측정치 보기(콘솔)를 참조하십시오.	2017년 2월 9일
새로운 서비스 및 가이드	이것은 이미지 분석 서비스인 Amazon Rekognition과 Amazon Rekognition 개발자 안내서의 최초 릴리스입니다.	2016년 11월 30일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조서의 AWS 용어집을 참조하십시오.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.