DOL-05

Amazon Rekognition で実現する 生体検出プロセス付き顔認証

Naoto Oiso

Aamazon Web Services
Solutions Architect

Naoto Oiso / 大磯 直人

Solutions Architect

業務

toC 向けの Web サービスを提供する会社様の
技術相談担当しています。

マイブーム

・マイホーム探し

趣味

・YouTube, 漫画, 技術書積読

好きな **AWS** サービス

Amazon Rekognition Face Liveness

不正ユーザーの対策に悩まれている方

不正ユーザーによる被害例 ~ マッチングアプリ ~

サービスユーザーの被害と、それによる企業としてのレピュテーションの被害

ロマンス詐欺 を狙う不正ユーザーの存在

ロマンス詐欺

ページ　ノート

ロマンス詐欺（ロマンスさぎ）は、主にインターネット上の交流サイトなどで知り合った海外の相手を言葉巧みに騙して、恋人や結婚相手になったかのように振る舞い、金銭を送金させる特殊詐欺の一種[2]。

国際ロマンス詐欺、国際恋愛詐欺などとも呼ばれる。代表的なものにはナイジェリア詐欺がある。

送金した場合、現状では事後に金銭を取り戻せるすべはほぼない[3]。

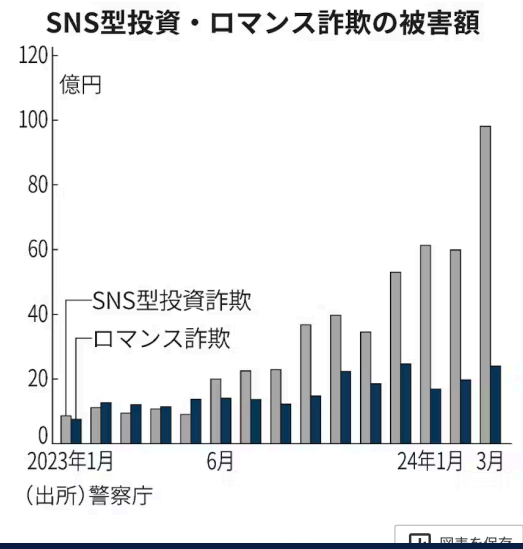
Wiki引用

SNS型投資・ロマンス詐欺が急増　1～3月4.5倍で279億円

事件・司法　＋フォローする

2024年5月16日 15:52 (2024年5月16日 16:06更新)

保存

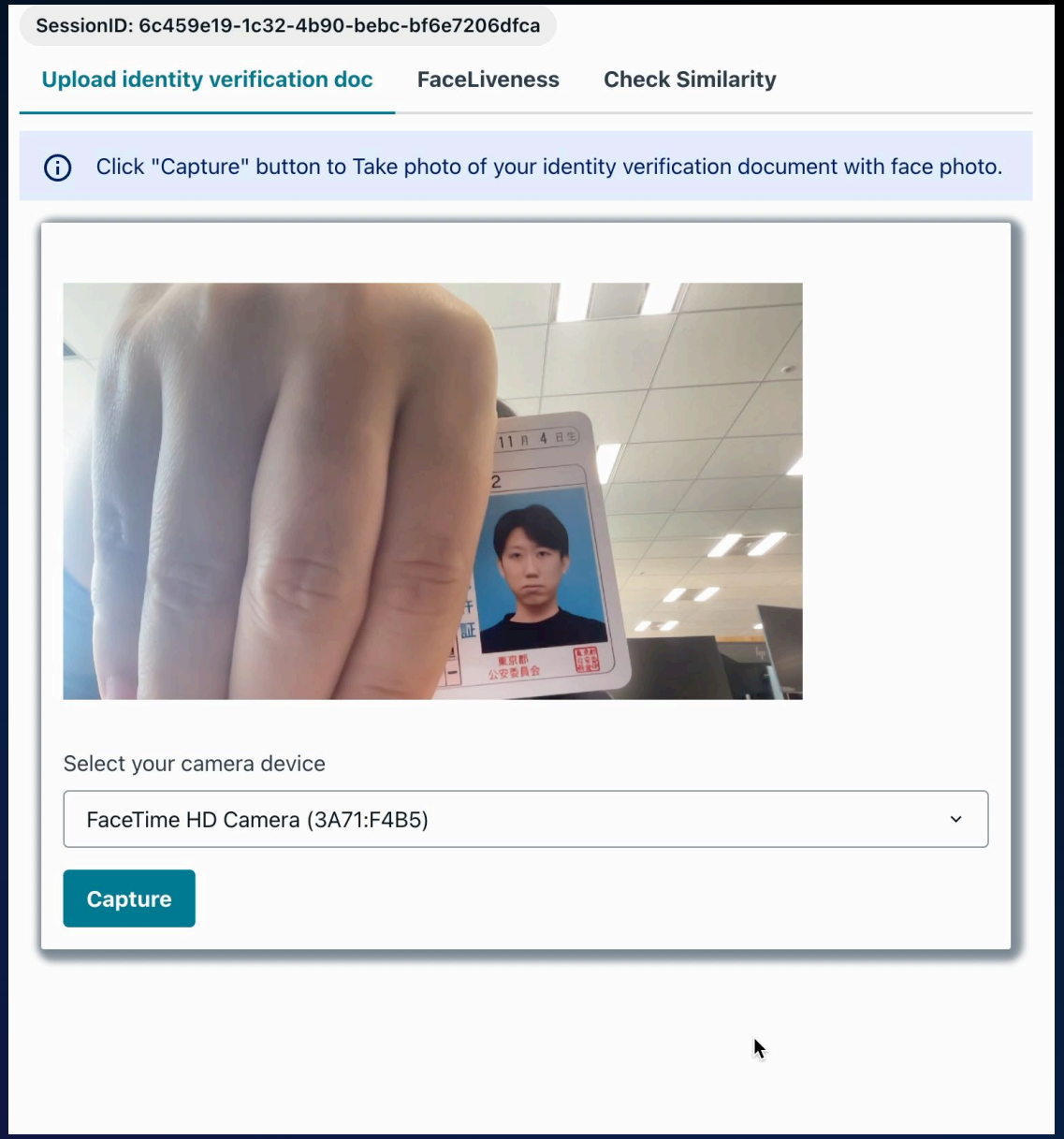


日経新聞引用

本プレゼンテーションの要約

不正ユーザーの対策でお困りの皆様に朗報です！

機械学習のスキルいらずで、
生体検出プロセス付きの顔認証が
約 $0.025 / 回 で実現できます！



Amazon Rekognition

Amazon Rekognition

深層学習をベースにした画像・動画認識 **AI** サービス

Amazon Rekognition Image

画像ファイル

Amazon Rekognition Video

動画ファイルや
ストリーミングビデオ

Amazon Rekognition の機能全体像 と 顔認証 で利用する機能

物体・シーン検出

顔検出・分析

顔の比較

有名人認識

コンテンツの
モデレーション

保護具検知

テキストの検出

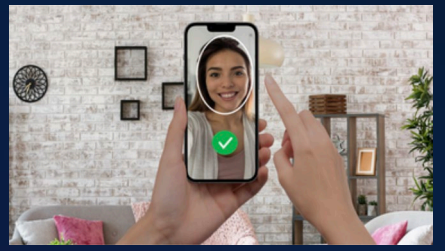
人物の動線追跡

動画のシーン分析

顔検索

動画ストリーミングの分析

カスタムラベル



生体検出

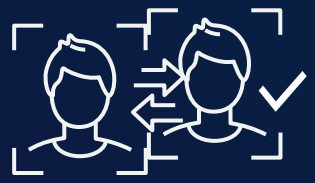
生体検出付き
オンライン顔認証 アーキテクチャ

Amazon Rekognition を利用した顔認証サンプルフロー

本人確認書類画像の顔検出 → 生体検出による不正確認 → 顔画像一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録

本人確認書類から本人画像を特定し、鮮明に検出できるかの確認

DetectFaces



生体検出を行い、被認証者が生身の人間であることを確認

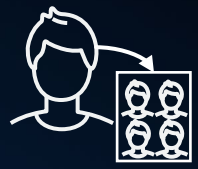
Face Liveness

本人確認書類の画像と生体検出した人物の画像との同一性を確認

CompareFaces

既存インデックスに、生体検出した人物の画像の検索を行い重複の有無確認

SearchFacesByImage

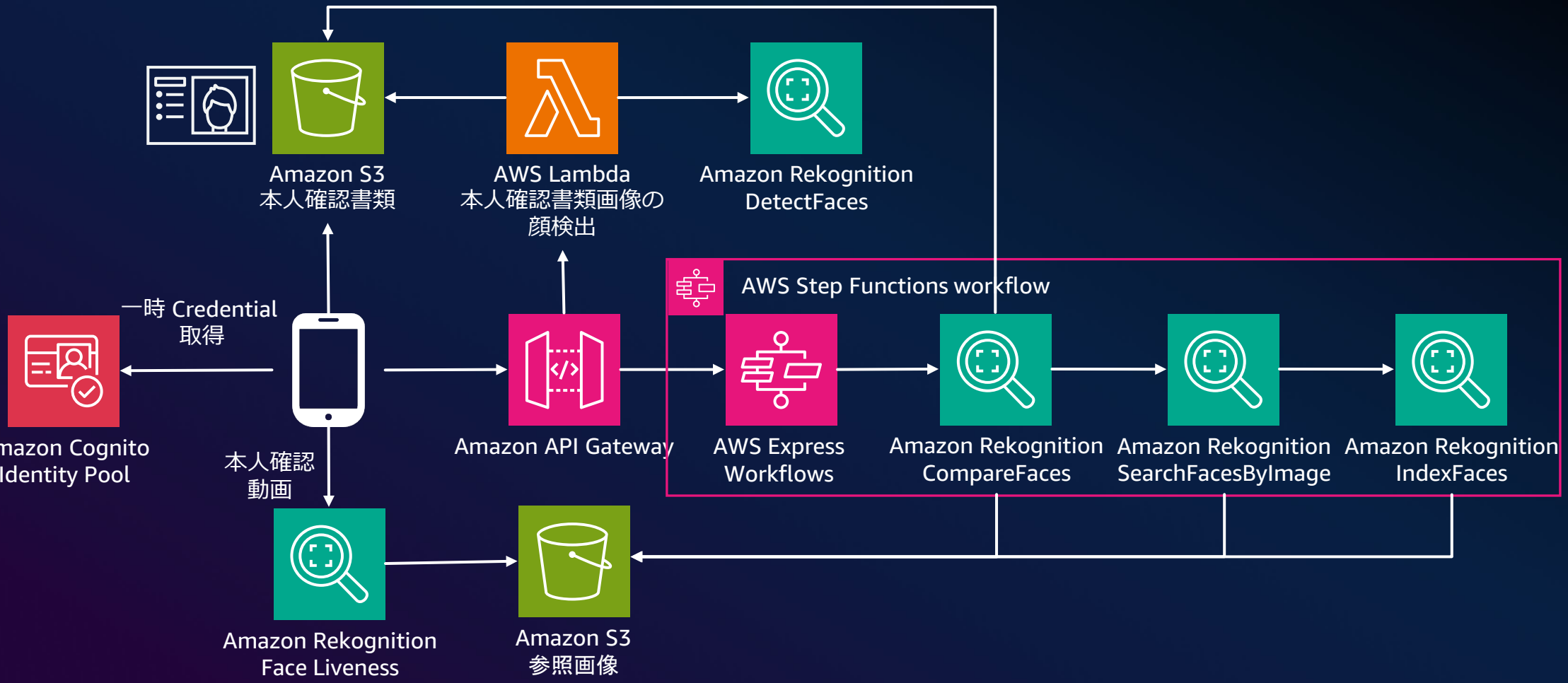


既存インデックスに、生体検出した人物の画像の登録を実施

IndexFaces

Amazon Rekognition を利用した 顔認証サンプルアーキテクチャ

本人確認書類画像の顔検出 → 生体検出による不正確認 → 顔画像一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録



Amazon Rekognition を利用した顔認証サンプルフロー

本人確認書類画像の顔検出 → 生体検出による不正確認 → 顔画像一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録

本人確認書類から本人画像を特定し、鮮明に検出できるかの確認

DetectFaces

生体検出を行い、被認証者が生身の人間であることを確認

Face Liveness

本人確認書類の画像と生体検出した人物の画像との同一性を確認

CompareFaces

既存インデックスに、生体検出した人物の画像の検索を行い重複の有無確認

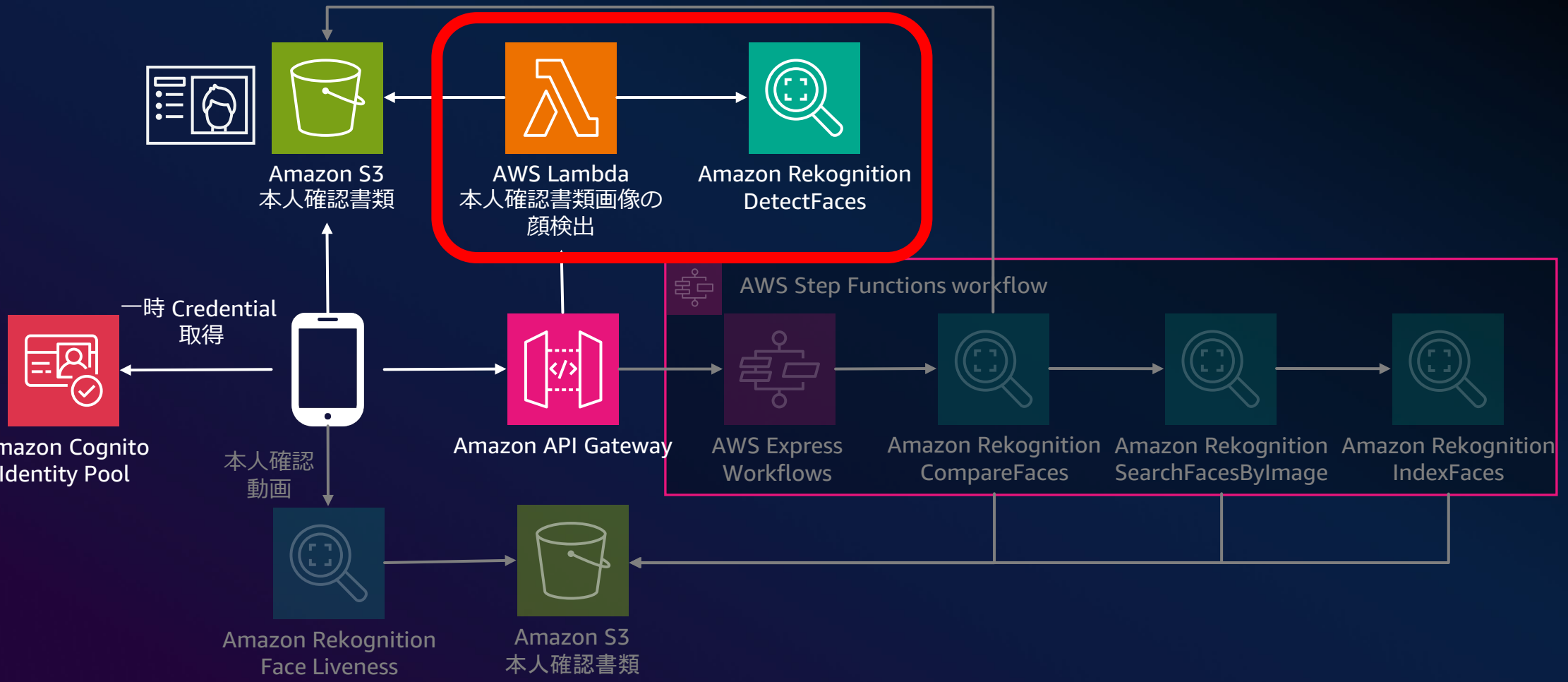
SearchFacesByImage

既存インデックスに、生体検出した人物の画像の登録を実施

IndexFaces

Amazon Rekognition を利用した 顔認証サンプルアーキテクチャ

本人確認書類画像の顔検出 → 生体検出による不正確認 → 顔画像一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録



本人確認書類での顔検出

```python
from PIL import Image

def lambda_handler(event, context):
        # Retrieve bucket name and file name from S3 event
        bucket = event['Records'][0]['s3']['bucket']['name']
        key = event['Records'][0]['s3']['object']['key']

        # Get the uploaded image
        response = s3_client.get_object(Bucket=bucket, Key=key)
        image_bytes = response['Body'].read()

        # 画像から顔を検出
        face_response = rekognition_client.detect_faces(
            Image={'Bytes': image_bytes},
            Attributes=['DEFAULT']
        )
        if face_response['FaceDetails']:
                # Use the information of the first detected face
                faceDetail = face_response['FaceDetails'][0]
                # Detect Faceのレスポンスから画像の顔の位置を抽出
                box = faceDetail['BoundingBox']
                image = Image.open(io.BytesIO(image_bytes))
                imgWidth, imgHeight = image.size
                left = imgWidth * box['Left']
                top = imgHeight * box['Top']
                width = imgWidth * box['Width']
                height = imgHeight * box['Height']
                # 画像の顔の箇所の切り抜き
                face_image = image.crop((left, top, left + width, top + height))
                # Convert the cropped face image to a byte array
                buffer = io.BytesIO()
                face_image.save(buffer, 'JPEG')
                buffer.seek(0)

                # Upload the cropped face image to S3
                face_key = 'faces/' + key.split('/')[-1]
                s3_client.put_object(Bucket=os.getenv('CROPED_BUCKET'), Key=face_key, Body=buffer, ContentType='image/jpeg')
```

```python
# 画像から顔を検出
face_response = rekognition_client.detect_faces(
    Image={'Bytes': image_bytes},
    Attributes=['DEFAULT']
)
```

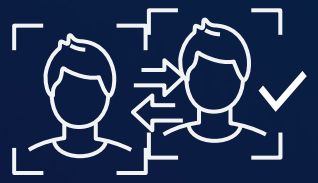
detect_faces メソッドを利用して、
画像の中から顔の検出を行う。

Amazon Rekognition を利用した顔認証サンプルフロー

本人確認書類画像の顔検出 → 生体検出による不正確認 → 顔画像一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録

本人確認書類から本人画像を特定し、鮮明に検出できるかの確認

DetectFaces



生体検出を行い、被認証者が生身の人間であることを確認

Face Liveness

本人確認書類の画像と生体検出した人物の画像との同一性を確認

CompareFaces

既存インデックスに、生体検出した人物の画像の検索を行い重複の有無確認

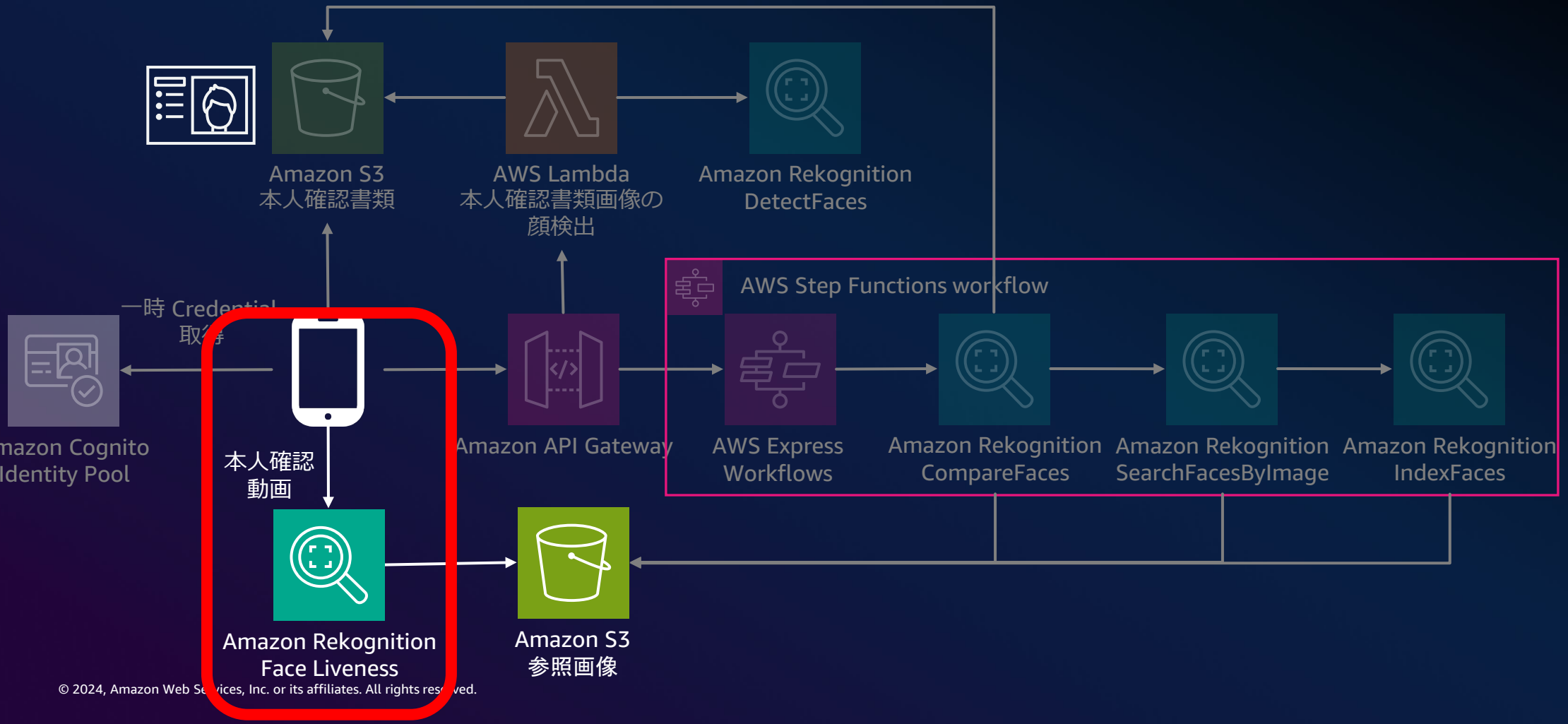
SearchFacesByImage

既存インデックスに、生体検出した人物の画像の登録を実施

IndexFaces

Amazon Rekognition を利用した 顔認証サンプルアーキテクチャ

本人確認書類画像の顔検出 → 生体検出による不正確認 → 顔画像一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録



Amazon Rekognition Face Liveness

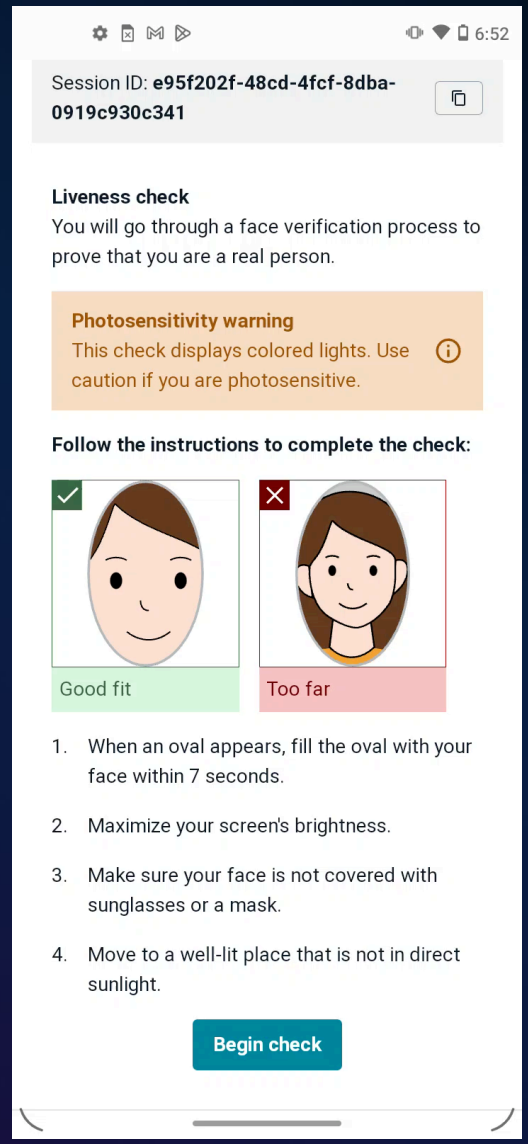
LIVENESS DETECTION による スプーフィング攻撃の検知及び緩和

リアルタイムの自撮りビデオを解析し、生体検出の信頼度スコア(0~100)を付与

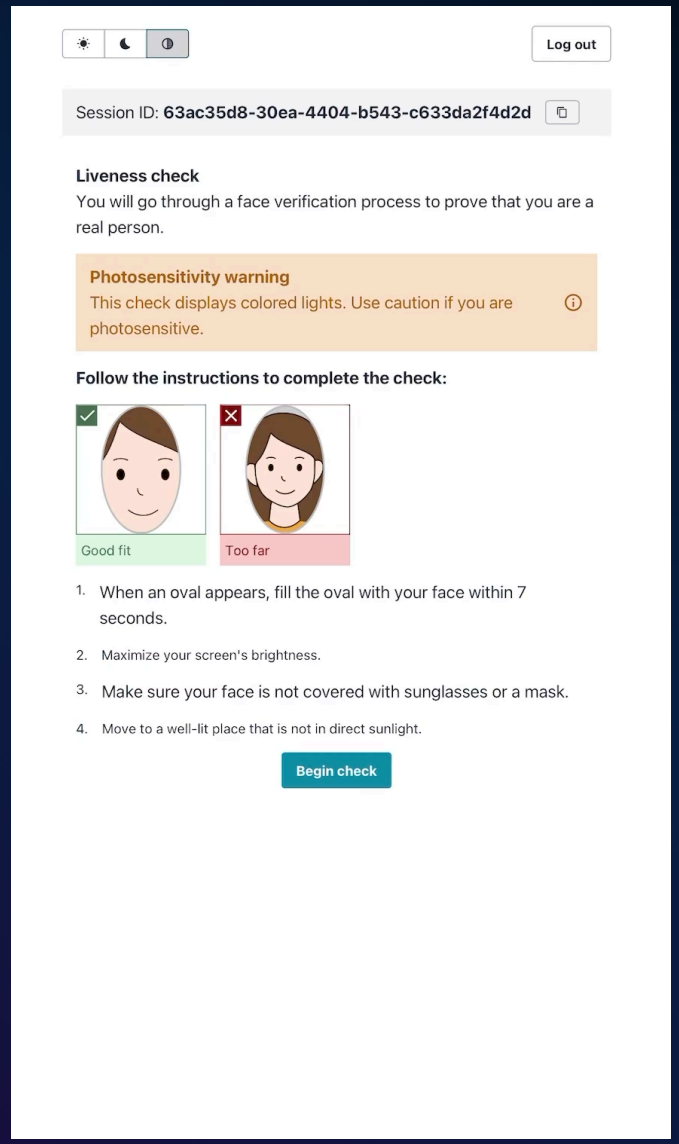
印刷された写真、デジタルスクリーン上の画像、3Dマスクなどの**プレゼンテーション攻撃を検出**

ディープフェイク、仮想カメラ、録画または操作されたビデオからの**インジェクション攻撃を検出**

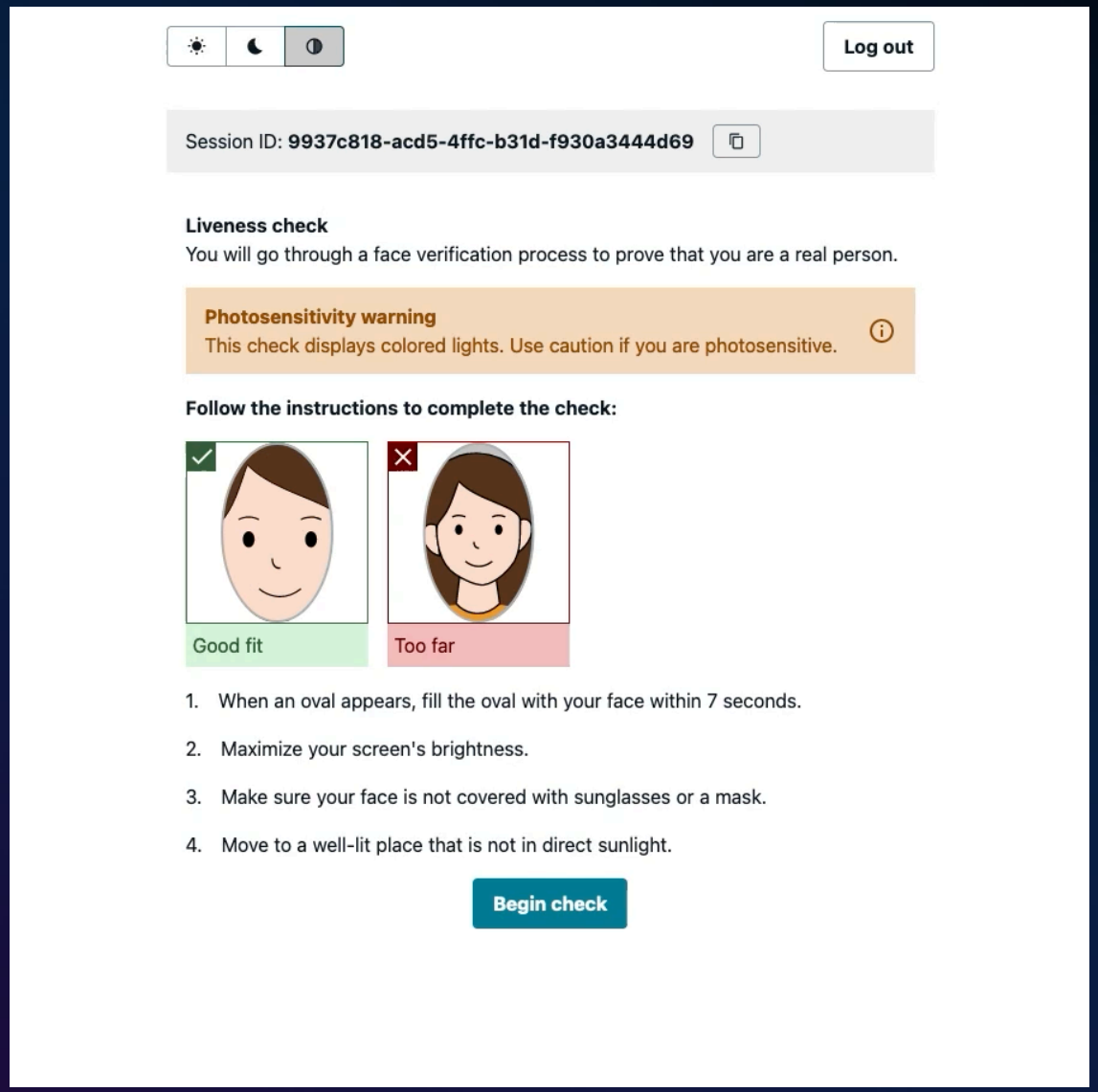
デモンストレーション： ディスプレイ プレゼンテーション攻撃



デモンストレーション： 3Dマスク プレゼンテーション攻撃



デモンストレーション： 合成ビデオ インジェクション攻撃



Face Liveness ~ Session ID 発行 (Server Side) ~

```python
import boto3
rek_client = boto3.client('rekognition')

def create_session():
    try:
        # Session ID 発行
        response = rek_client.create_face_liveness_session()
        return response.get("SessionId")
    except FaceLivenessError:
        ~
```

create_face_liveness_session メソッドを利用して、
Face Liveness のセッションを開始する

Face Liveness ~ UI表示 (Client Side React) ~

```
import React from "react";
import { FaceLivenessDetector } from '@aws-amplify/ui-react-liveness';

 # Session ID を受取り、生体検出 UI 表示
function FaceLiveness({ faceLivenessAnalysis, sessionid }) {
        return (
                <FaceLivenessDetector
                  sessionId={sessionid}
                  region={region}
                  displayText={dictionary[language]}
                  onAnalysisComplete={handleAnalysisComplete}
                  onError={(error) => {
                  console.error(error);
                }}
                />
        );
}
```

生体検出を行うための UI を表示する UI コンポーネントの
FaceLivenessDetector にSession IDを渡す

Face Liveness ~ 結果取得 (Server Side) ~

```python
import boto3
rek_client = boto3.client('rekognition')

def get_session_results(event):
    try:
        # Session ID を受取り、生体検出 の結果確認
        response = rek_client.get_face_liveness_session_results(SessionId=event['sessionid'])
        return response.get("SessionId")
    except FaceLivenessError:
        ~
```

get_face_liveness_session_results メソッドに Session ID を渡して、
Face Liveness のスコアを取得する。

Amazon Rekognition を利用した顔認証サンプルフロー

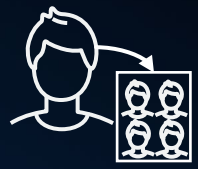
本人確認書類画像の顔検出 → 生体検出による不正確認 → 本人確認書類一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録

本人確認書類から本人画像を特定し、鮮明に検出できるかの確認

生体検出を行い、被認証者が生身の人間であることを確認

本人確認書類の画像と生体検出した人物の画像との同一性を確認

既存インデックスに、生体検出した人物の画像の検索を行い重複の有無確認



既存インデックスに、生体検出した人物の画像の登録を実施

DetectFaces

Face Liveness

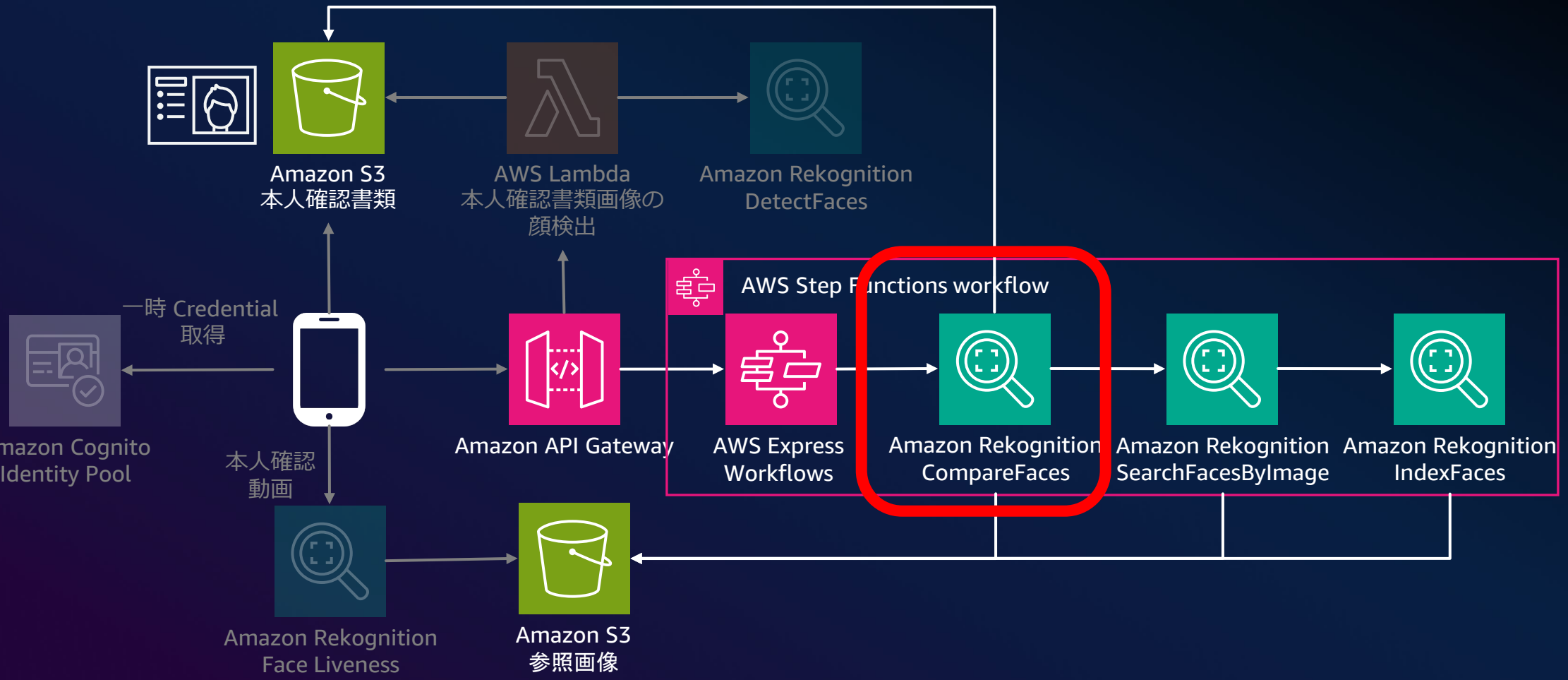
CompareFaces

SearchFacesByImage

IndexFaces

Amazon Rekognition を利用した 顔認証サンプルアーキテクチャ

本人確認書類画像の顔検出 → 生体検出による不正確認 → 本人確認書類一致確認 → 既存ユーザーとの重複確認 → インデックスに新規登録



本人確認書類の顔 と Face Liveness の顔の比較

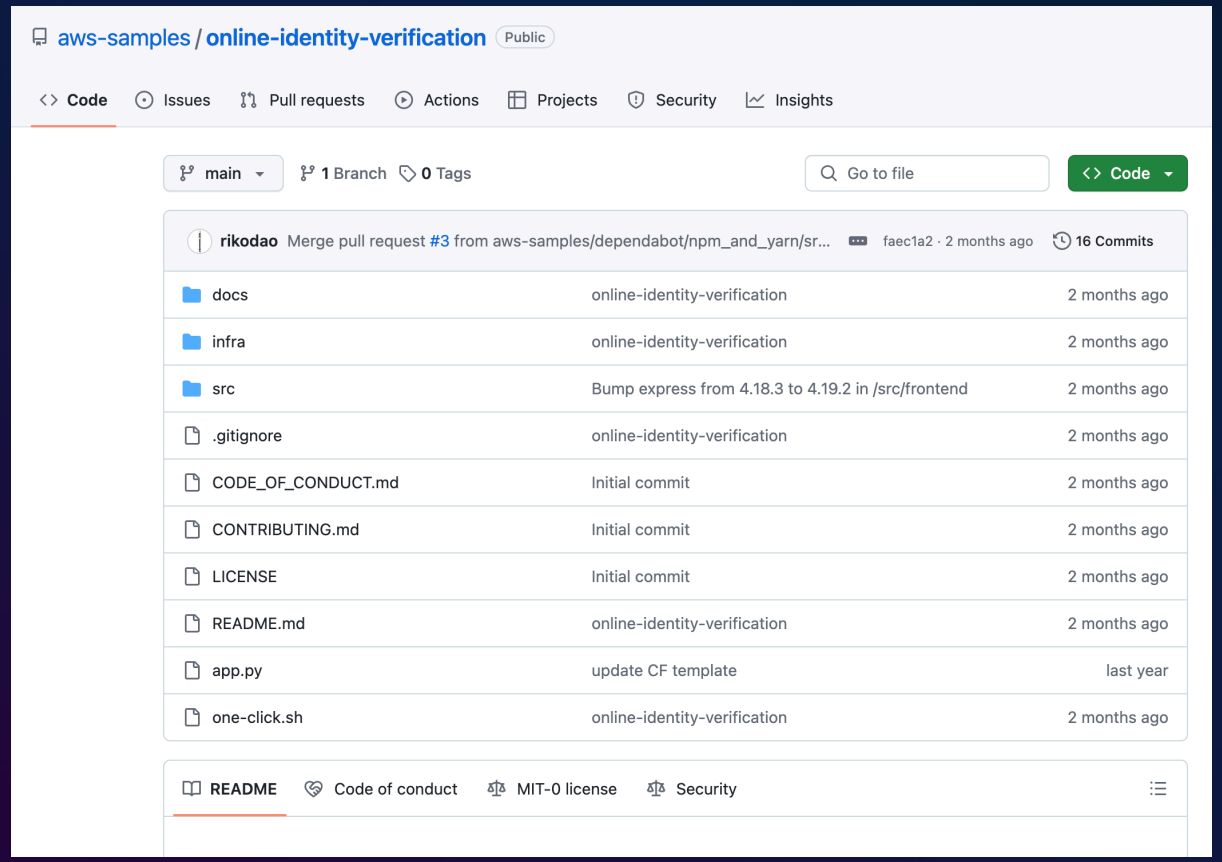
```
import boto3
rek_client = boto3.client('rekognition')

def compare_faces(self, sessionid, similarity_threshold=50):
    try:
        # 本人確認書類から得られた顔画像
        croppedImageKey = 'faces/' + sessionid + ".jpg"
        # Face Liveness で得られた顔画像
        FaceLivenessImageKey = "referenceImage/" + sessionid + '.jpg'

        # 本人確認書類の顔 と # Face Liveness で得られた顔の比較
        response = rek_client.compare_faces(
            SourceImage={'S3Object': {'Bucket': crop_face_bucket_name, 'Name': croppedImageKey}},
            TargetImage={'S3Object': {'Bucket': faceliveness_face_bucket_name, 'Name': FaceLivenessImageKey}},
            SimilarityThreshold=similarity_threshold
        )
        return response
    except Exception as e:
     ~
```

compare_faces メソッドを利用して、２つの顔の比較を行う

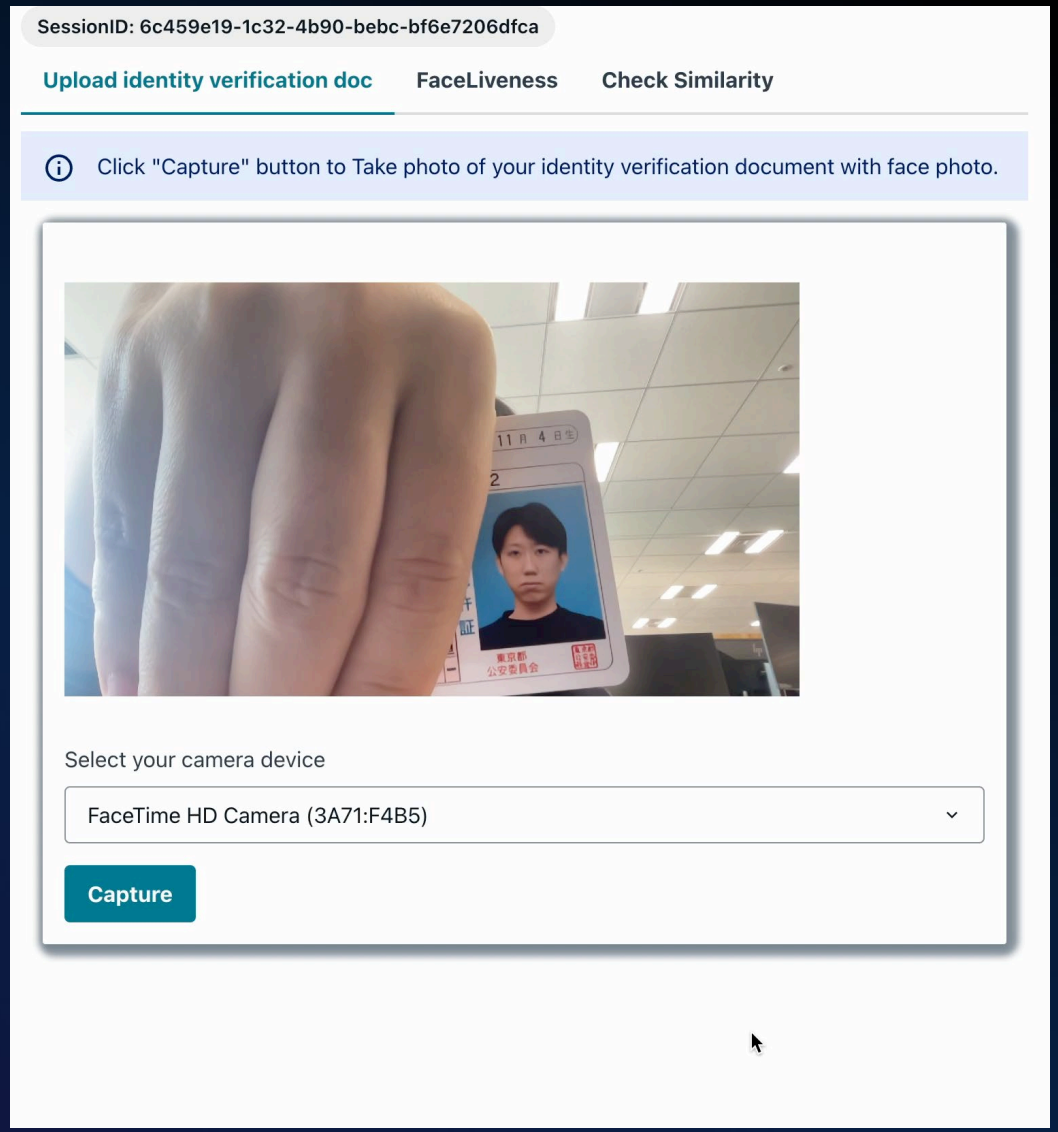
Next Action

デモアプリの GitHub リポジトリ

皆様のお手元でもお試しいただけます



https://github.com/aws-samples/online-identity-verification



Thank you!

Naoto Oiso
naotoiso@amazon.co.jp

デモアプリリポジトリのQRコード

JAPAN | JUNE 20, 2024

aws SUMMIT