Face Detection and Image Analysis Using Amazon Rekognition

Project Report

14-Dec-2020

Team Members: Shawn Gonsalves

Nikhil Bora

Yumi Park

Bradley Stukas

**Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Deployed Stack Successfully | 12/2 | CS446-Stack Deployment Failure(Code Directory) | 1.0 |
| Implemented DetectFacesAPI with advanced features | 12/4 | DetectFacesAPI could detect the faces but without any additional parameters | 2.1 |
| Added Rekognize\_CelebrityAPI | 12/5 | Could not differentiate between Celebrities and a real person | 2.2 |
| Added ImageModeration\_API | 12/7 | Could not detect if the image provided contained any explicit content | 2.3 |
| Added BlackListImage Feature | 12/10 | Wasn’t able to check if the same image has had already been uploaded | 2.4 |

**Table of Contents**

[**Deliverables**](#_gjdgxs) **4**

[**Executive Summary**](#_30j0zll) **5**

[**Project Overview**](#_1fob9te) **6**

[Problem Statement](#_3znysh7) 6

[Project Team](#_2et92p0) 6

[**Cloud Computing Aspects**](#_tyjcwt) **7**

[**Software**](#_3dy6vkm) **8**

[AWS services utilized](#_1t3h5sf) 8

[Amazon Rekognition](#_8h4c7xvszo8f) 8

[AWS S3](#_8h4c7xvszo8f) 9

[AWS Lambda](#_8h4c7xvszo8f) 9

[AWS CloudFormation](#_8h4c7xvszo8f) 10

[AWS Step Functions](#_8h4c7xvszo8f) 10

[AWS Elasticsearch](#_8h4c7xvszo8f) 10

[System Architecture and Pipeline](#_4d34og8) 11

[User Interface Overview](#_2s8eyo1) 12

[**Use Cases and tests**](#_17dp8vu) **16**

[**References**](#_3rdcrjn) **24**

[**Glossary**](#_26in1rg) **25**

# Deliverables

Code Repo (if any) (URL): <https://github.com/shawngonsalves/CS446_AmazonRekognition>

Presentation Slides (URL): <https://drive.google.com/drive/folders/1FUCbh8hUTjCXRv5zKEktC8Z0PneHckDu?usp=sharing>

Presentation/Demo Video (URL): <https://drive.google.com/drive/folders/1FUCbh8hUTjCXRv5zKEktC8Z0PneHckDu?usp=sharing>

# Executive Summary

We researched on one of the powerful AWS Rekognition services for face detection and analyzing images for different parameters; for example, whether or not a face is detected, or if it matches certain criteria such as wearing sunglasses, the facial expression, or if it matches the features of another face in the registry. For this case study, we were able to implement AWS Rekognition APIs in conjunction with other AWS services such as S3 buckets and AWS Lambda to create a system that can apply pre-existing AWS Rekognition filters to filter out photos that meet specific criteria.

This project was a good learning experience to practically apply cloud services, how powerful the services are, and how to use them together in tandem.

# Project Overview

## Problem Statement

The purpose of our project was to do a case study on the Amazon Rekognition software. This report serves as both a record of our research, as well as a documentation of our implementation of Rekognition and any other AWS services utilized.

## Project Team

Name of the project:

|  |  |  |
| --- | --- | --- |
| **Name of the Team member** | **Responsibility** | **Contribution %** |
| Shawn Gonsalves | Amazon Rekognition implementation, researched on AWS CloudFormation, editor | 25% |
| Nikhil Bora | Project Manager, Researched on DetectFacesAPI, AWS StepFunction and AWS ElasticSearch | 25% |
| Yumi Park | Researched on AWS DetectFacesAPI, IndexFacesAPI and AWS Lambda. | 25% |
| Bradley Stukas | RecognitionCelebrity, ModerationLabelsAPI, researcher | 25% |

# Cloud Computing Aspects

Cloud computing refers to the idea that rather than having all of the resources you need to run your operation at your physical location you can access resources on-demand from offsite locations using the internet. In cloud computing, microservices are a concept where instead of grouping all of your services together into one system, which would be a monolithic system, you separate the systems into different services that can be scaled independently. This helps avoid having to scale the whole system together in order to increase the power of one part of the system. Distributed computing is a system where a single software system has different parts of itself shared across many different computers that run as one system. This practice can make it a lot easier to scale up systems once they need more processing power.

Amazon Rekognition is part of the larger AWS system of microservices supplied by Amazon. This system also includes the other Amazon microservices we are using such as ec2 and s3. All of these systems are provided as separate services so that projects can scale easily in only the parts that are needed. For example, if we were to have allowed other outside sources to have access to running our step functions and get results, then that part could scale to the increased demand without the parts earlier in our system to have to scale with it.

# Software

## AWS services utilized

### Amazon Rekognition

The focus of the project is implementing Amazon Rekognition, which is a cloud-based Software as a service computer vision platform that eases the process of adding images and video analysis to your applications using highly scalable, deep learning technology that requires no machine learning expertise to use. Amazon Rekognition can identify objects, people, text, scenes, and activities in images and videos, as well as detect any inappropriate content. It also provides highly accurate facial analysis and facial search capabilities that you can use to detect, analyze, and compare faces for a wide variety of user verification, people counting, and public safety use cases.

Our implementation of Rekognition is to use as a facial recognition filter that blocks pictures that meet certain criteria. These criteria are implemented as Rekognition API, which reads in a JSON file and returns the evaluation result. Further details on the process are outlined in section 5.2

The following are the Amazon Rekognition APIs we used in our project:

1. DetectFaces: Detects faces in a given input image and provides us the face details as the output such as no face detected, face detected with mouth open or eyes closed, face detected with wearing sunglasses, etc.
2. DetectLabel: Detects if the given input image contains an image of a user-provided label such as flower, landscape or nature, etc
3. CreateCollection: Creates a collection inside the AWS region which is used by the IndexImage operation to add and store images.
4. IndexFaces: Detects and adds input images to a specified collections.
5. ListFaces: Lists all images present in a specific collection.
6. RecognizeCelebrities: Detects whether the given input image is of some recognized celebrity.
7. DetectModerationLabel: Detects faces in an input image and checks for whether the image contains explicit or suggestive content such as nudity.

### AWS S3

Amazon S3 is a Simple Storage Service that provides developers and IT-Teams with Secure, durable, and highly-scalable Object Storage through a web service interface. It can be used for several purposes like to store and protect any amount of data for a Company’s websites, mobile applications, enterprise applications, etc. It provides easy-to-use management features so we can organize our data and configure access controls to meet our organizational and business requirements. It guaranteed 99.99% durability, availability. We can use it to store and retrieve data at any time and from anywhere in the world.

We use AWS S3 in our project to create buckets inside an AWS region and to store input images to be further processed through our Rekognition filter. An S3 bucket is a public cloud storage resource. Most importantly, an S3 bucket name requires a globally unique ID as the namespace is shared across all AWS accounts.

### AWS Lambda

AWS Lambda is an event-driven, serverless computing platform that lets you run code without provisioning or managing servers. It runs code and charges for computation only when necessary and automatically scales as project demand increases. Using Lambda delegates the resource management, deployment, logging, and other administrative activities related to running code to the service, which automates the process and leaves the user with only the responsibility of developing code.

We used this service in our project as it runs our project code to events and automatically manages the underlying compute resources for us.

When we deploy our project repository from the local machine to the AWS cloud, all the code(.py) files will get stored inside the AWS Lambda service. Inside AWS Lambda code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload. In our project, when we upload input images to S3, StepFunction will trigger AWS Lambda to process data.

### AWS CloudFormation

With AWS CloudFormation, users can manage related resources and related dependencies as a single unit called a stack. Users create, update, and delete a collection of resources by creating, updating, and deleting stacks. All the resources in a stack are defined by the stack’s AWS CloudFormation template. Users can provision and manage stacks across multiple AWS regions and AWS accounts. This service can invoke Lambda functions asynchronously with an event and provide a library cfn-response that handles sending the response. It then adds the library to the deployment package that it creates for the function.

In our project, we used AWS CloudFormation console and a YAML template to create a stack consisting of IAM roles, Lambda functions, and a state machine.

### AWS StepFunctions

AWS StepFunctions is a feature that manages AWS Lambda functions and other AWS services to build applications. StepFunctions provides a graphic console that visually displays the application workflow as a series of event-driven steps. It is versatile and can be utilized in multiple ways, but our implementation utilizes it for function orchestration, in which a group of Lambda functions is provided and are run in a specific order. The output of one Lambda function is provided to the next.

In our implementation, StepFunctions is utilized to manage and display each of the Rekognition API filters (run through AWS Lambda as lambda functions) as a multi-step process. This allows us to see each filter as a step as it is applied to a given input and where in the process it fails, if it fails. The use cases in section 6 demonstrate the visualization of the process provided by StepFunctions.

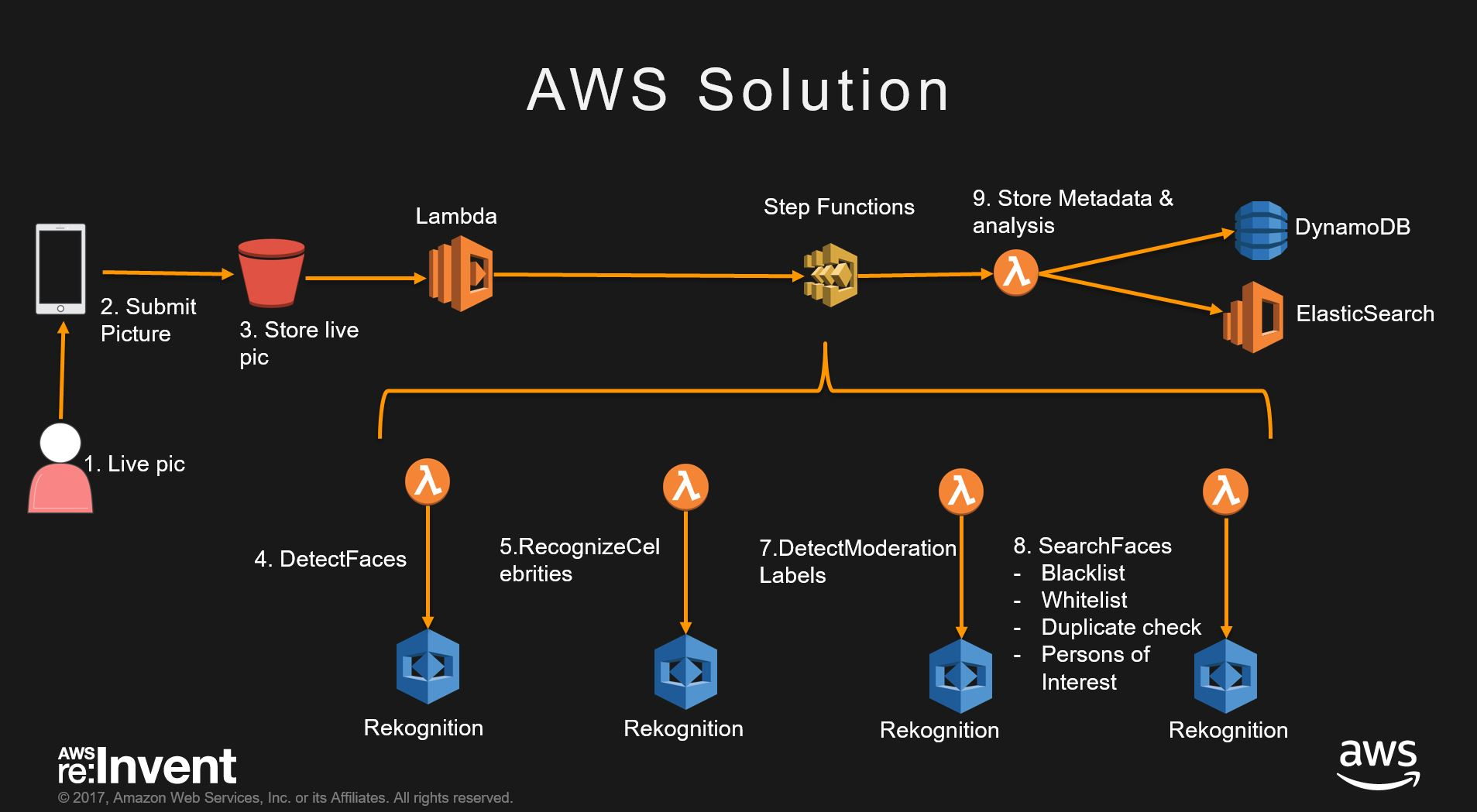
### AWS Elasticsearch

Elasticsearch is an open-source, RESTful search and analytics engine built on Apache Lucene. Data is sent in the form of JSON documents and Elasticsearch stores the original document and creates a searchable reference to the document within the cluster’s index. These stored documents can be retrieved and visualized using Kibana.

Our implementation utilizes Elasticsearch to store images which pass all checks and filters into the cluster. It is then referenced during future entry filters from the SearchFaces filter to avoid any other duplicate images from being entered into the cluster. Images which match any on the cluster are removed.

## System Architecture and Pipeline

The below diagram depicts the workflow to utilize Amazon Rekognition for this implementation. It is a basic image filtering system that pulls from a container and stores passing images.



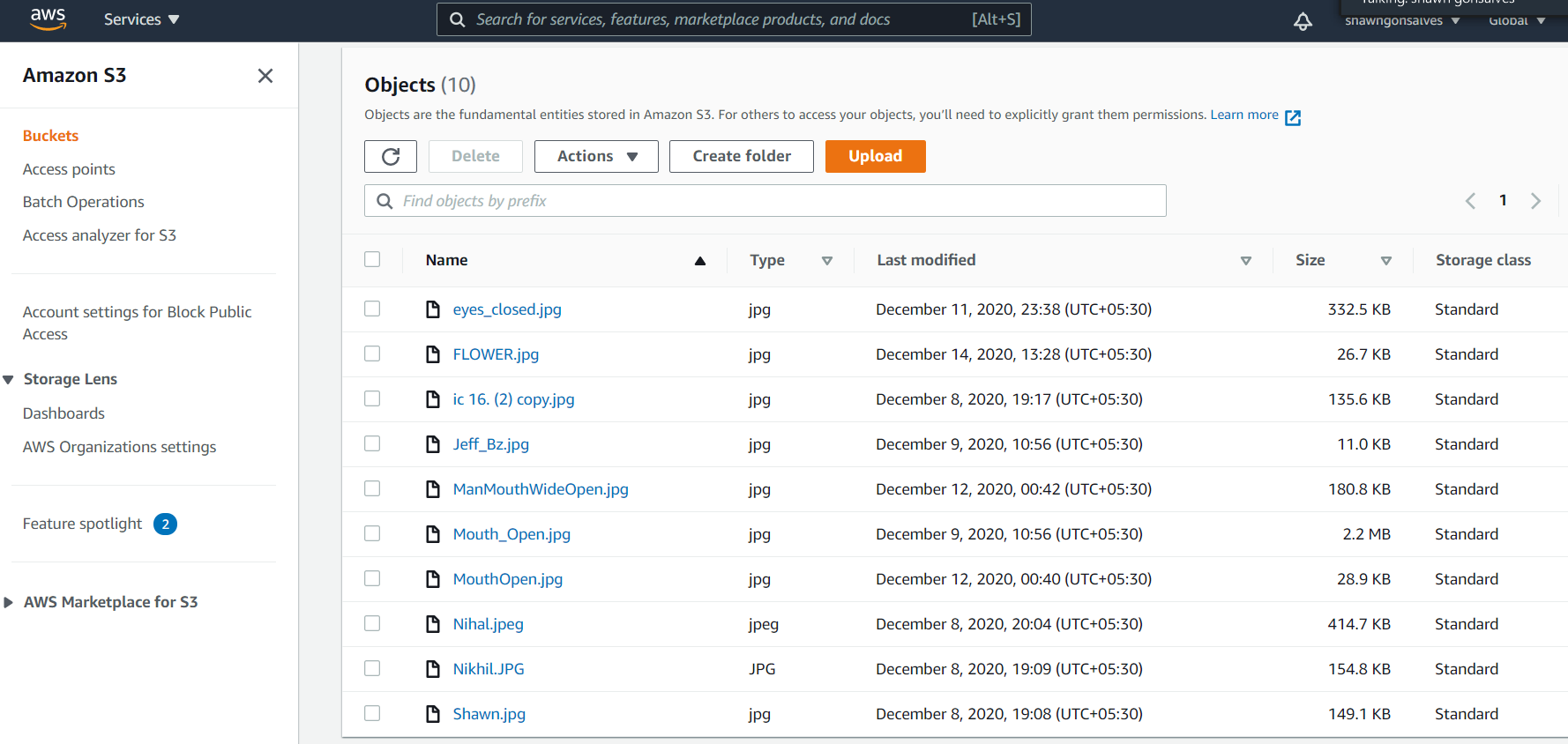
*Diagram provided by aws-samples*

Step-by-step process:

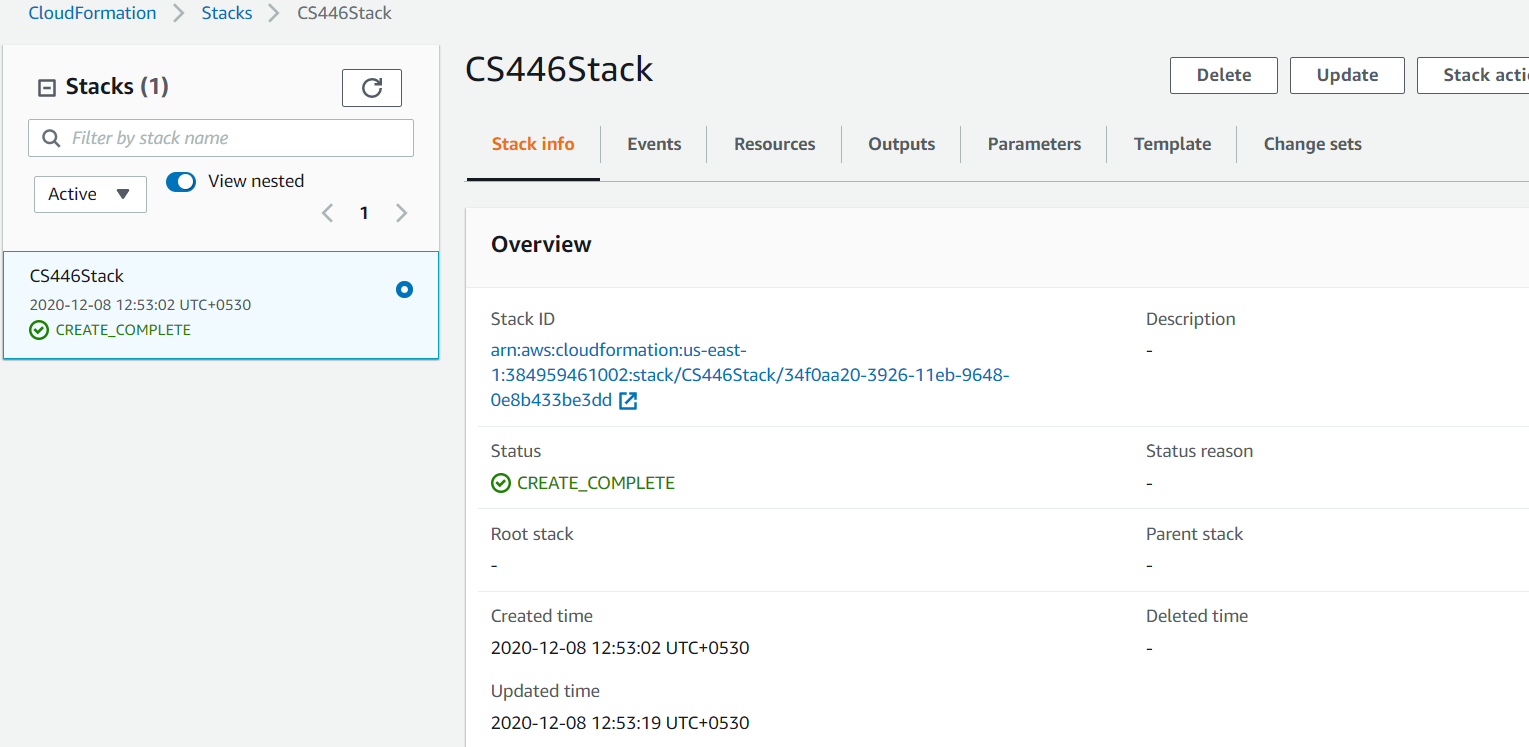
1. A user takes a picture
2. User submits the picture from their device
3. The picture is stored into an S3 bucket. At this point, AWS Lambda and AWS Step Functions are used to implement Rekognition APIs.
4. DetectFaces will detect faces and filter out faces with features that are not allowed. These features include wearing sunglasses, having their eyes closed, facing the wrong direction, or having their mouth open.
5. RecognizeCelebrities filters out any faces which are recognized as celebrities.
6. DetectModerationLabels filters any image which contains explicit or suggestive content.
7. SearchFaces will filter out any faces which are defined on a predefined blacklist or duplicates.
8. Results are logged into Amazon Elasticsearch, and displayed using Kibana.

## User Interface Overview

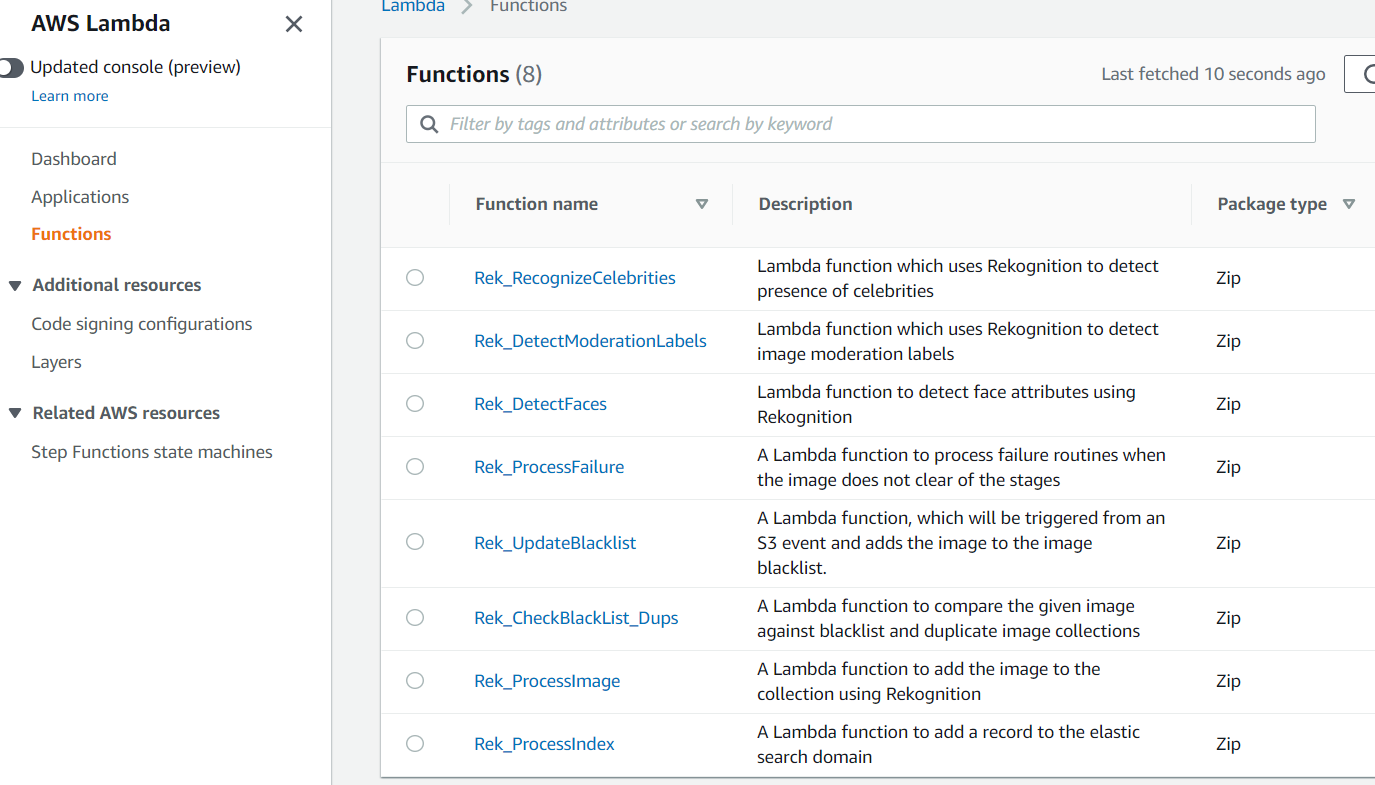
Input Images stored in the Amazon S3 Bucket



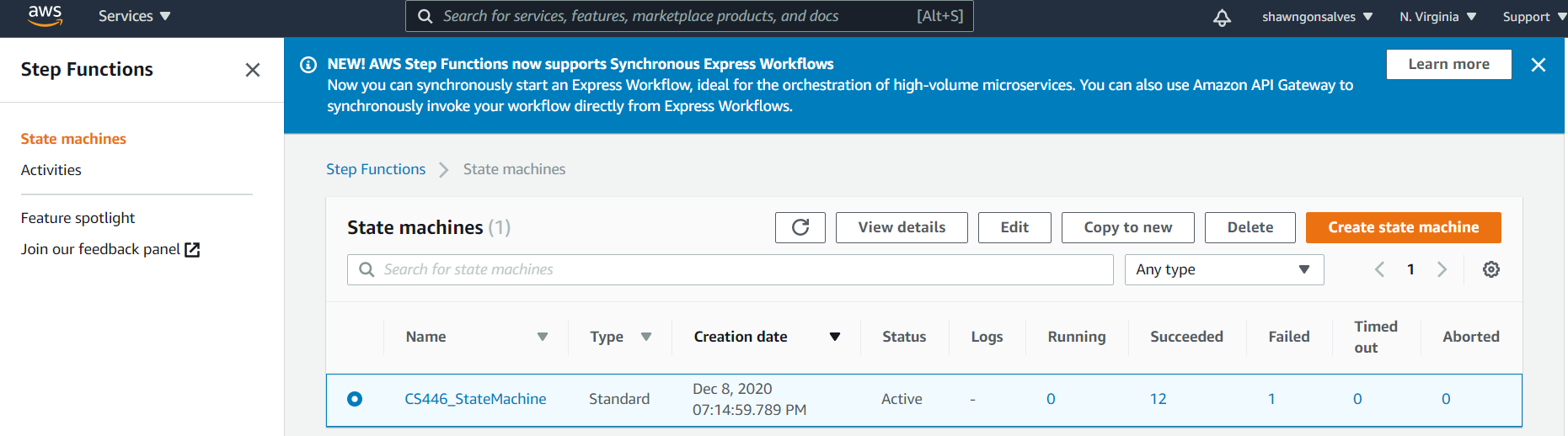
When we deployed our project repository from our local machine to AWS cloud, The Stack is created inside the AWS CloudFormation.



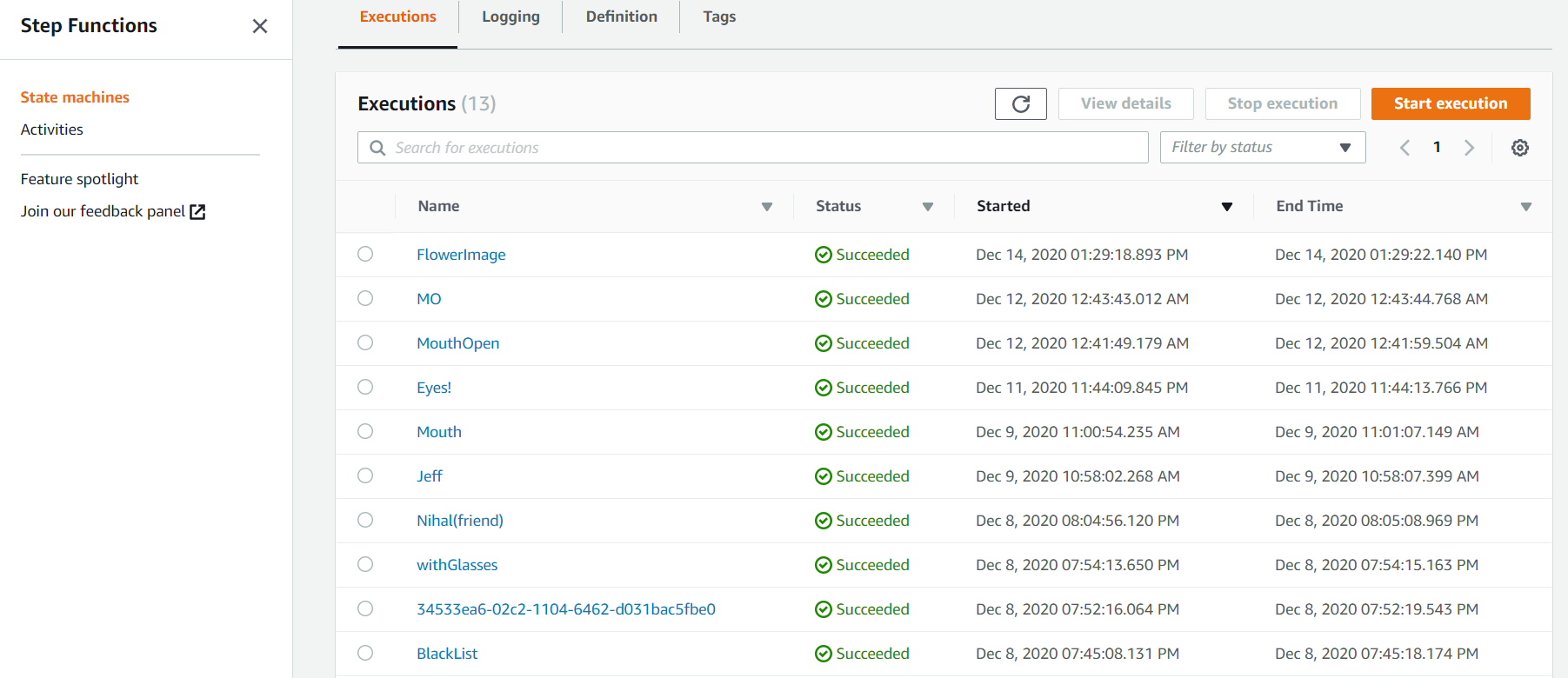
AWS Lambda runs our project code to events and automatically manages the underlying computing resources for us. In our project, when we upload input images to S3, StepFunction will trigger AWS Lambda to process data.



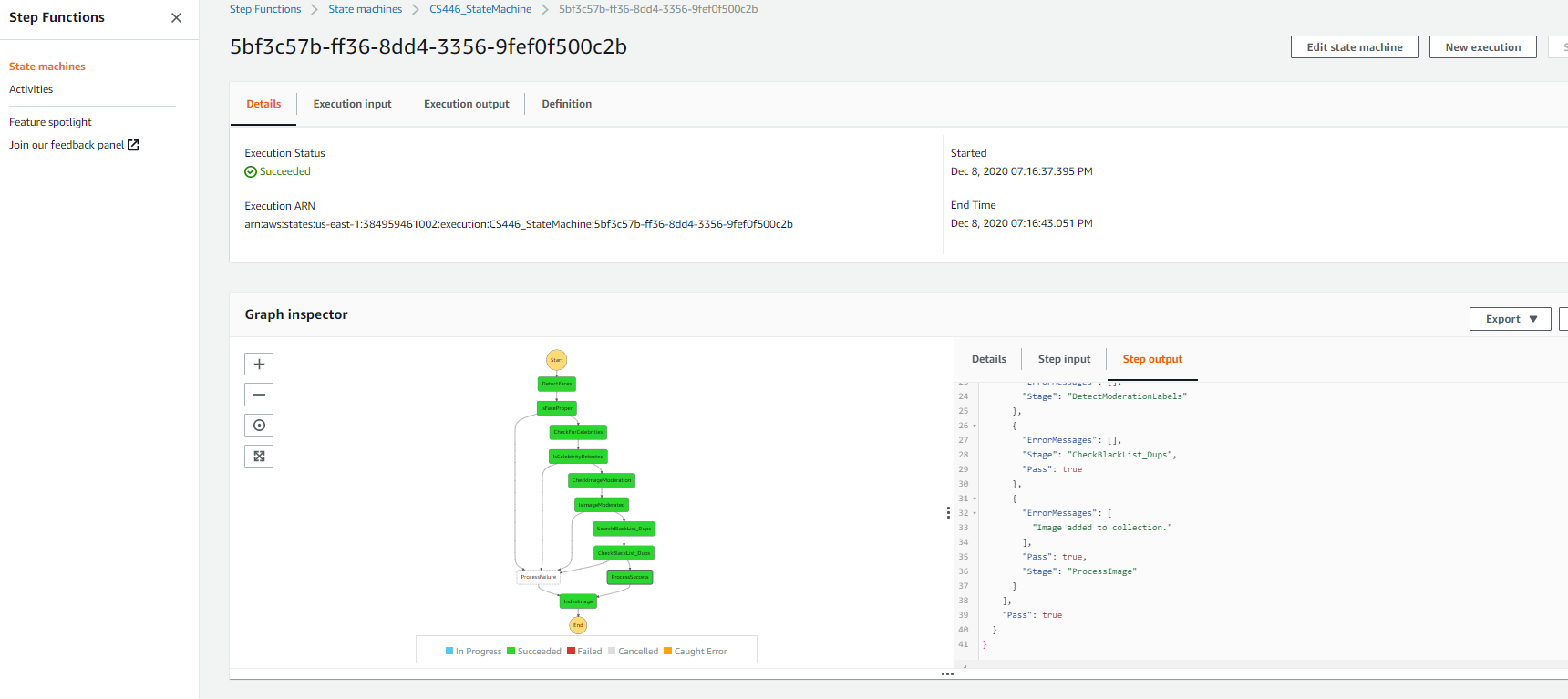
In the StepFunction the CS446\_StateMachine is created at the time Stack creation inside the CloudFormation.



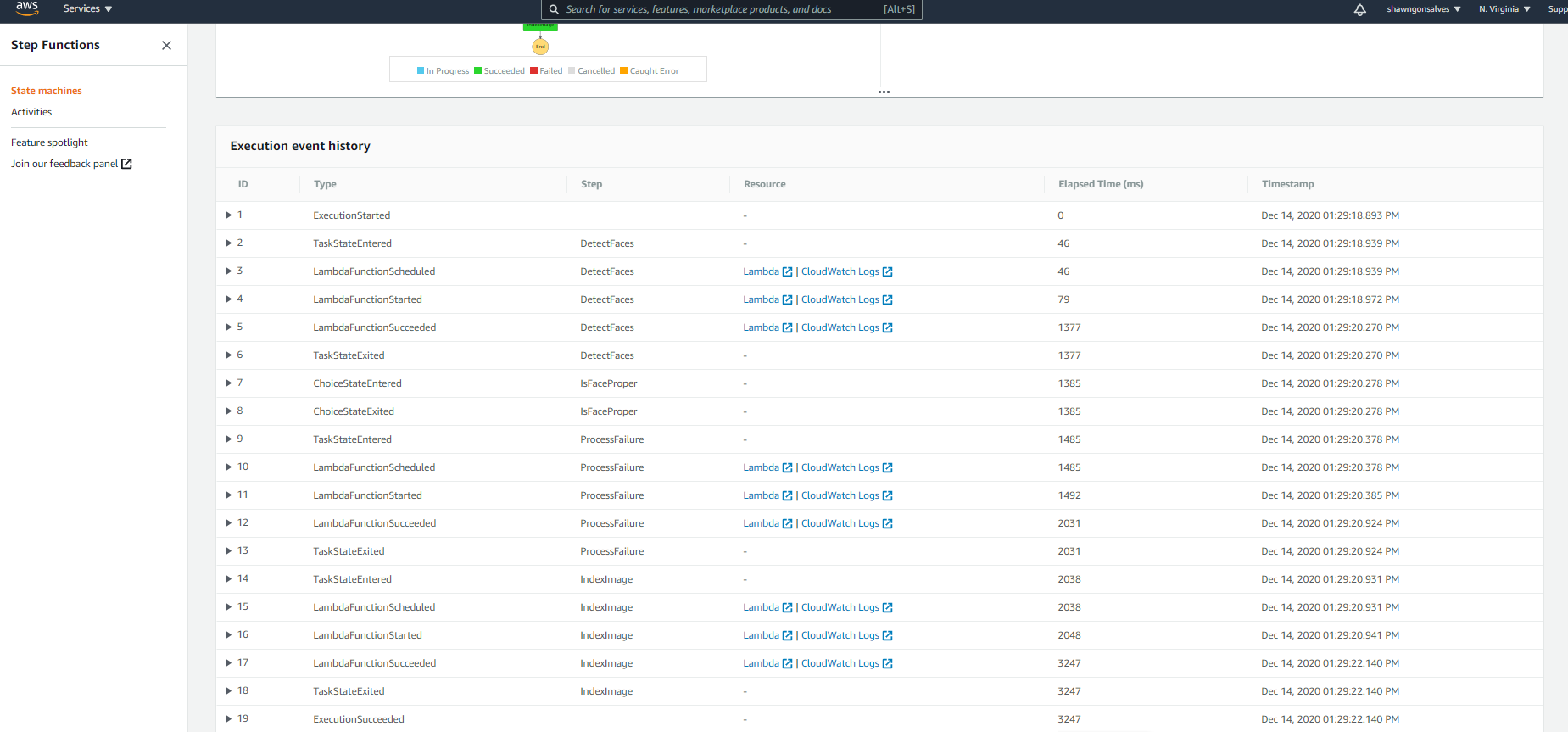
We executed our test cases for our project inside the StepFunctions service.



Following is the snapshot of a successfully executed test case



The below snapshot shows the Event History from the time the Execution started till the Execution succeeded.



# Use Cases and tests

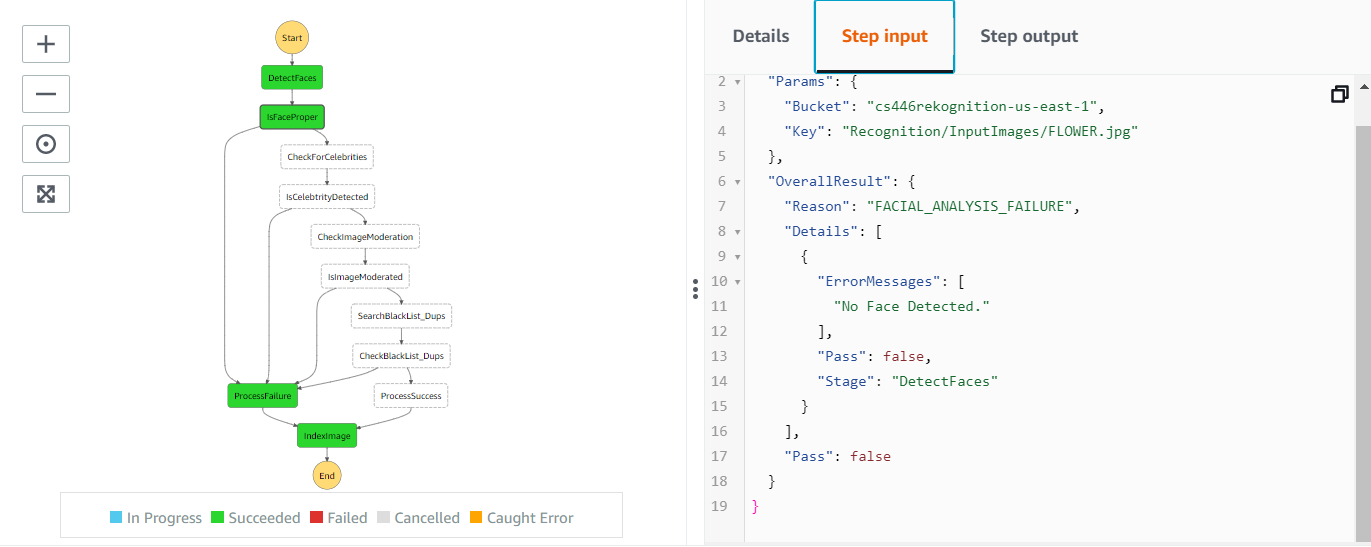
The below section depicts all the use cases we conducted to test every parameter and Rekognition filter used in the project.

1. **Adding a picture without a face.**

Input Image



After Execution



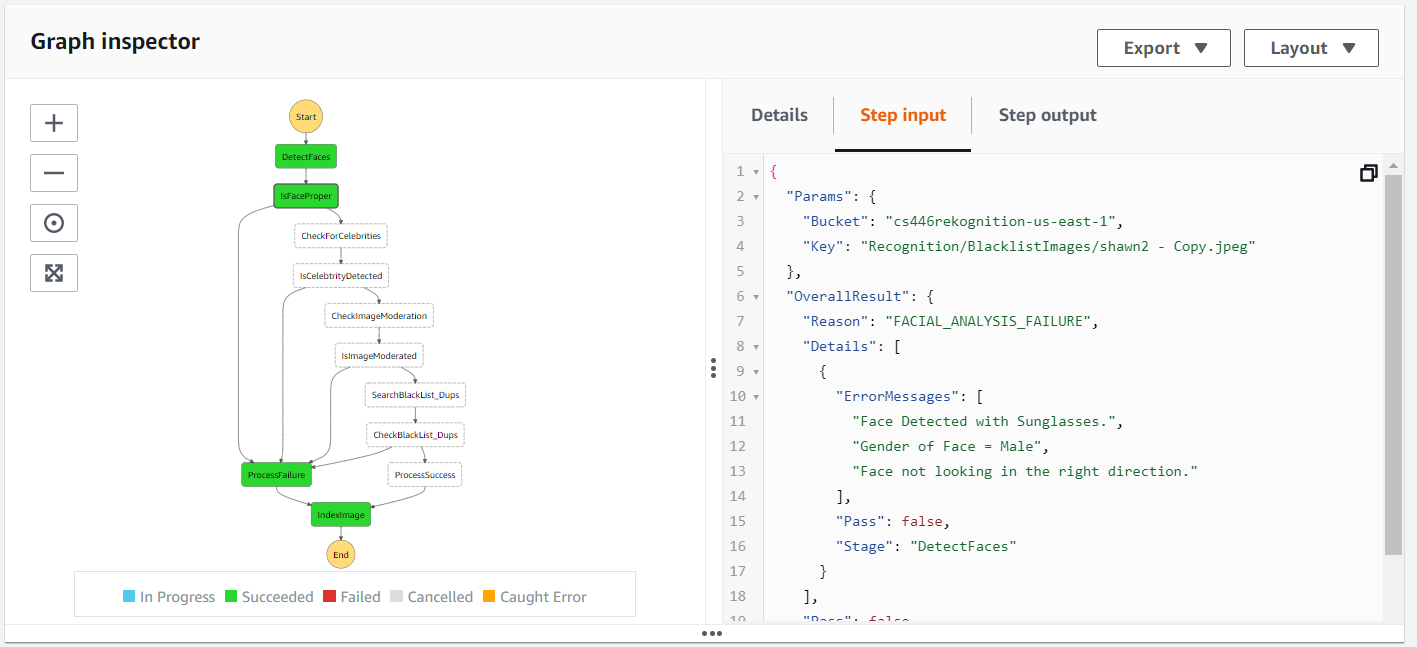
Since we passed an Image of Flower, the Lambda function for DetectFacesAPI gave us an error message: “No face Detected”.

1. **Looking in the wrong direction**

Input Image



After Execution



The above image gave us two error messages:

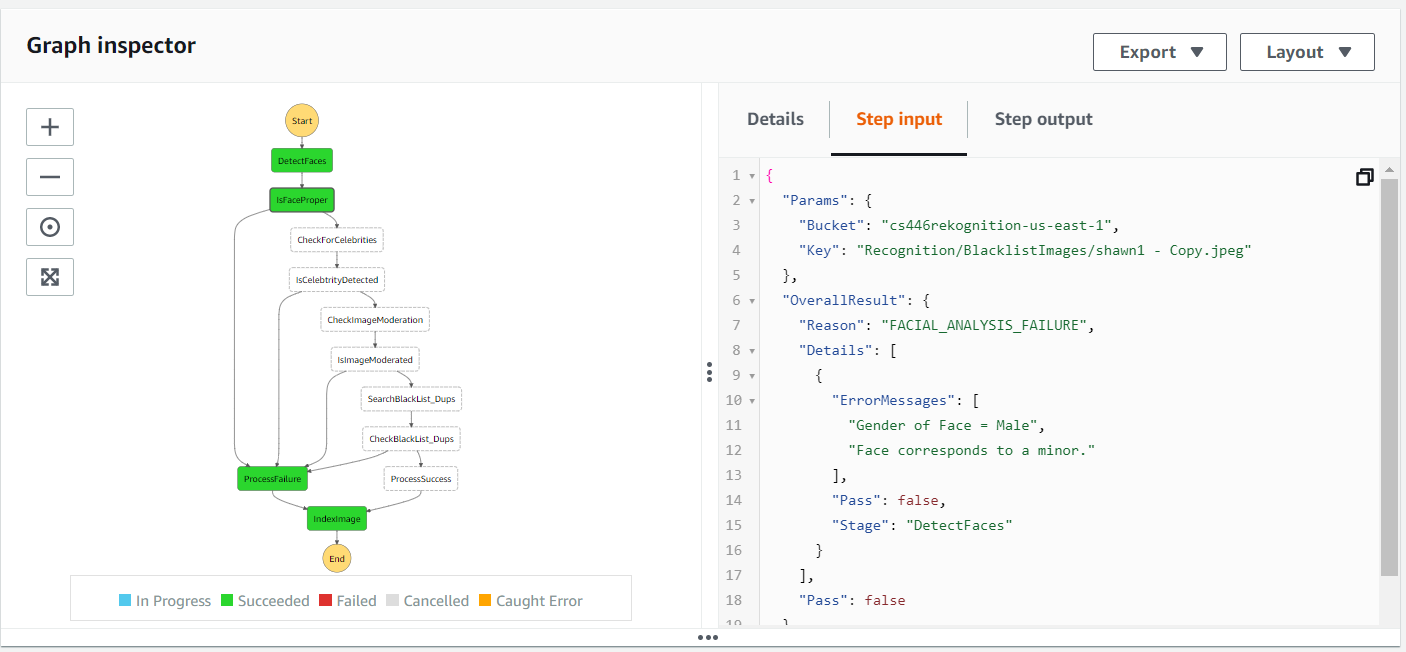
1. Face detected with sunglasses

2. Face not looking in the right direction.

1. **Correspondence to a Minor**



Input Image



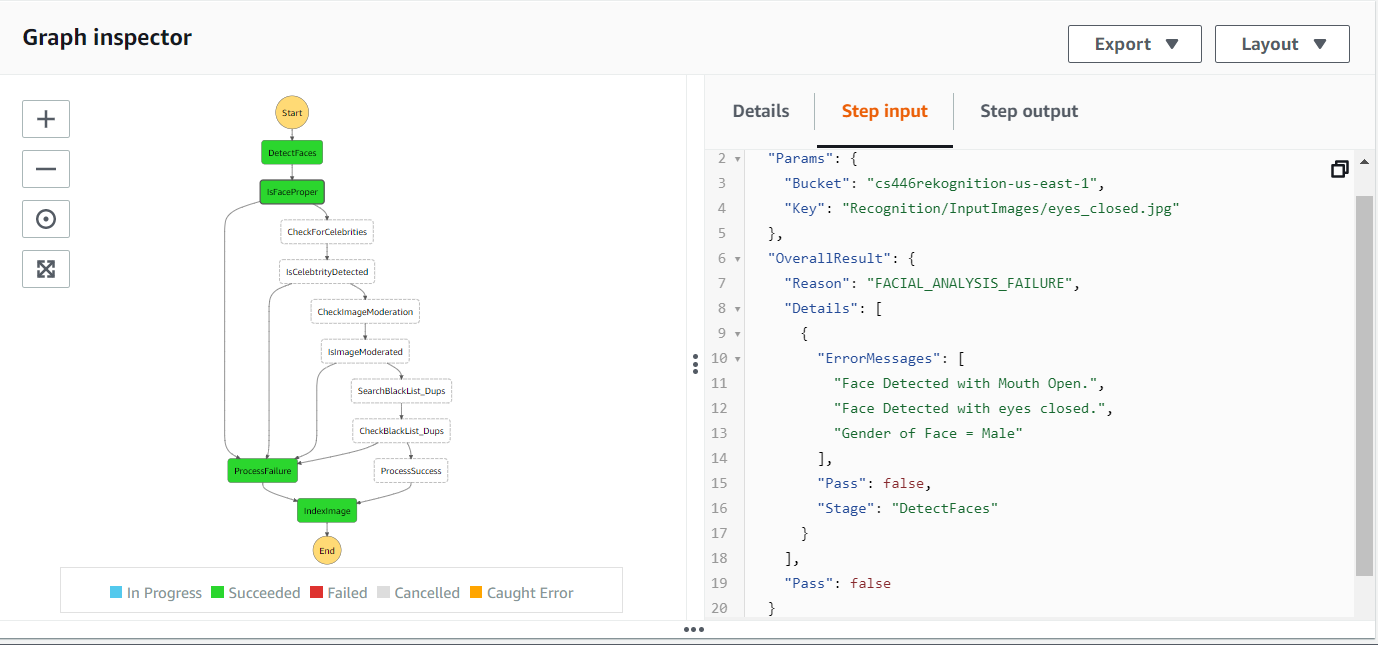
After Execution

The above image sent to the Step function as a key to its directory in the S3 bucket gave us an error: “Face corresponds to a Minor”

1. **Face Detection with Eyes Closed**



Input Image



After Execution

The image provided gave us two errors:

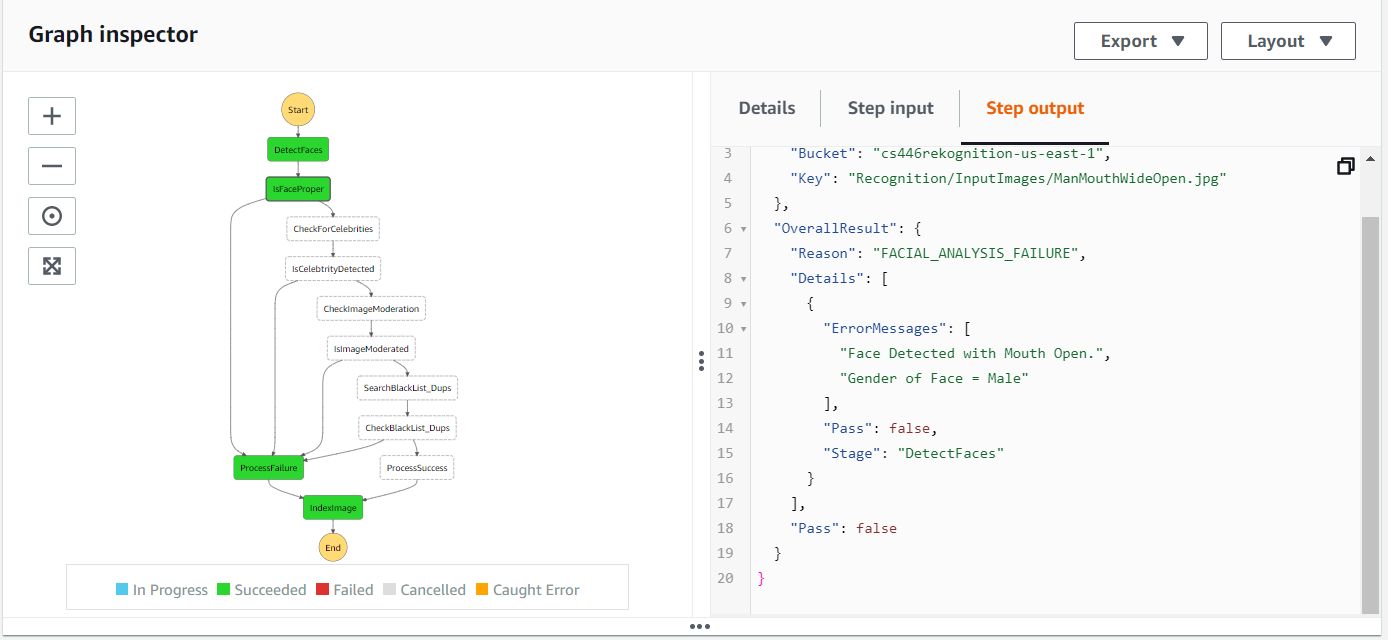
1. Face was detected but with Mouth Open.

2. Face Detected with Eyes Closed”

1. **Face Detected with Mouth open**



Input Image



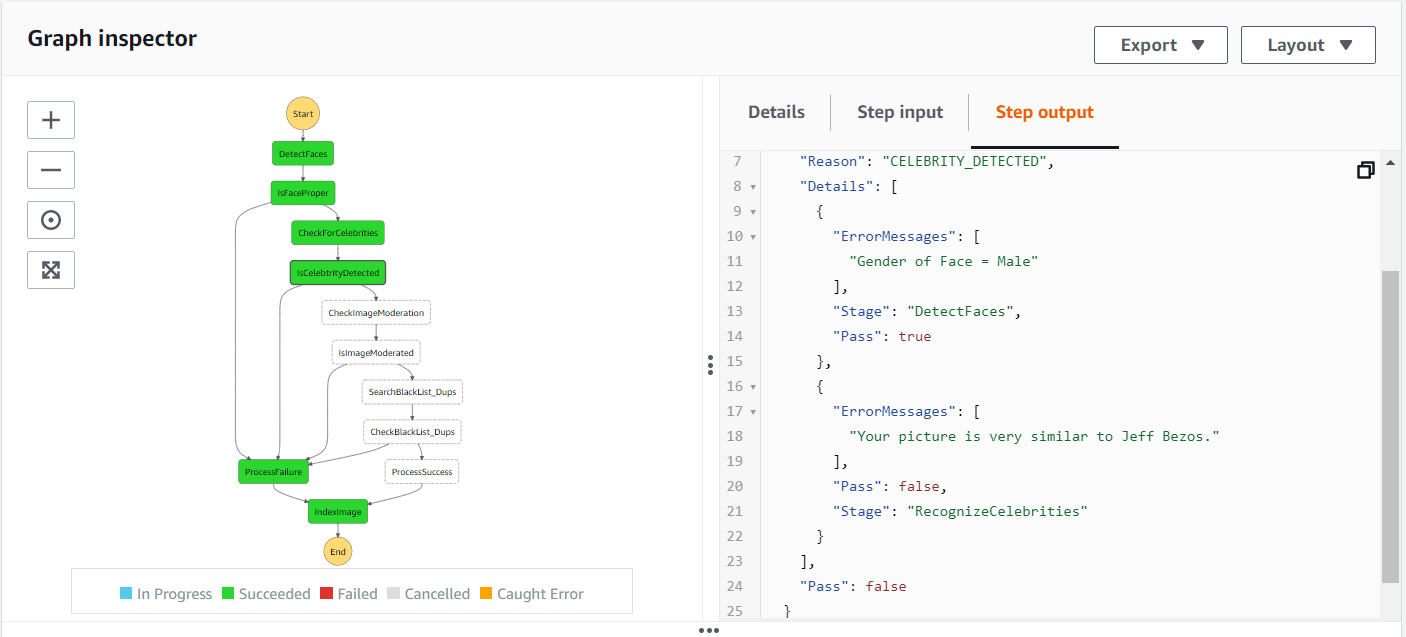
After Execution

The above image gave us an error: “Face detected with mouth Open”.

1. **Image of a Recognized Celebrity**



Input Image



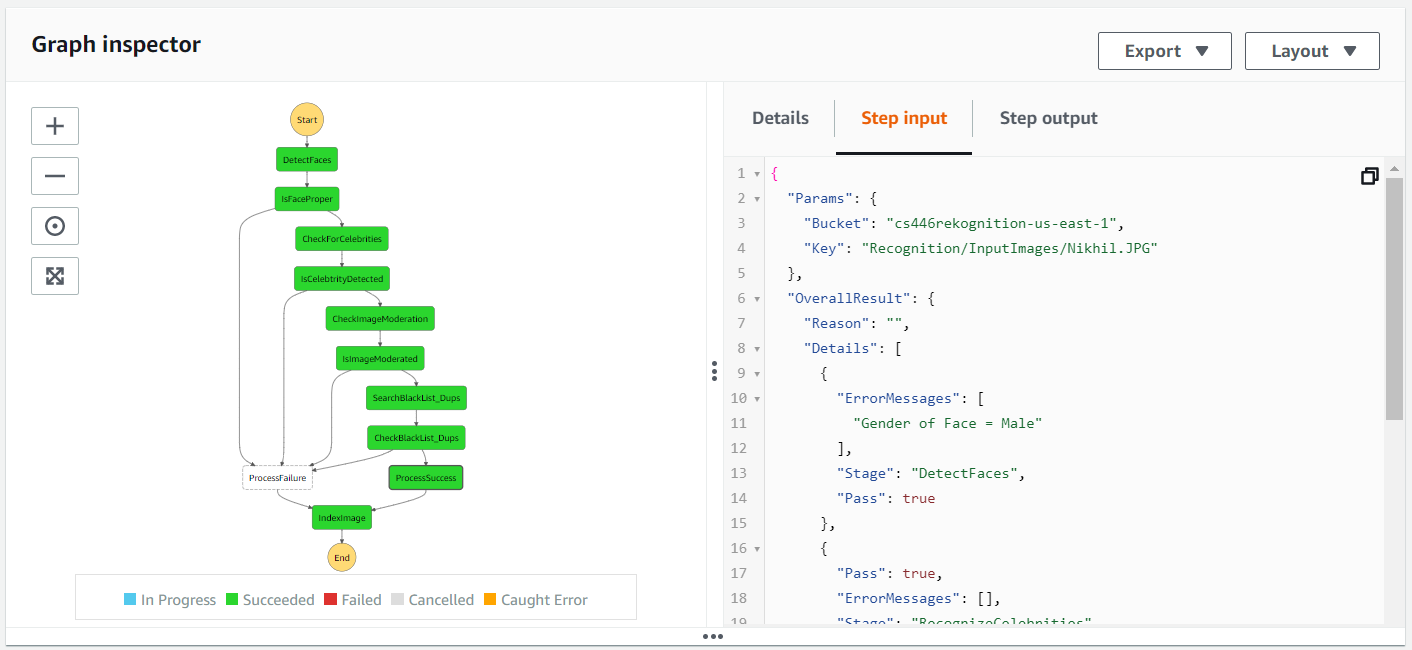
After Execution

The uploaded image gave us an error saying “The picture uploaded is similar to Jeff Bezos”

1. **Successfully Passing all parameters**



Input Image



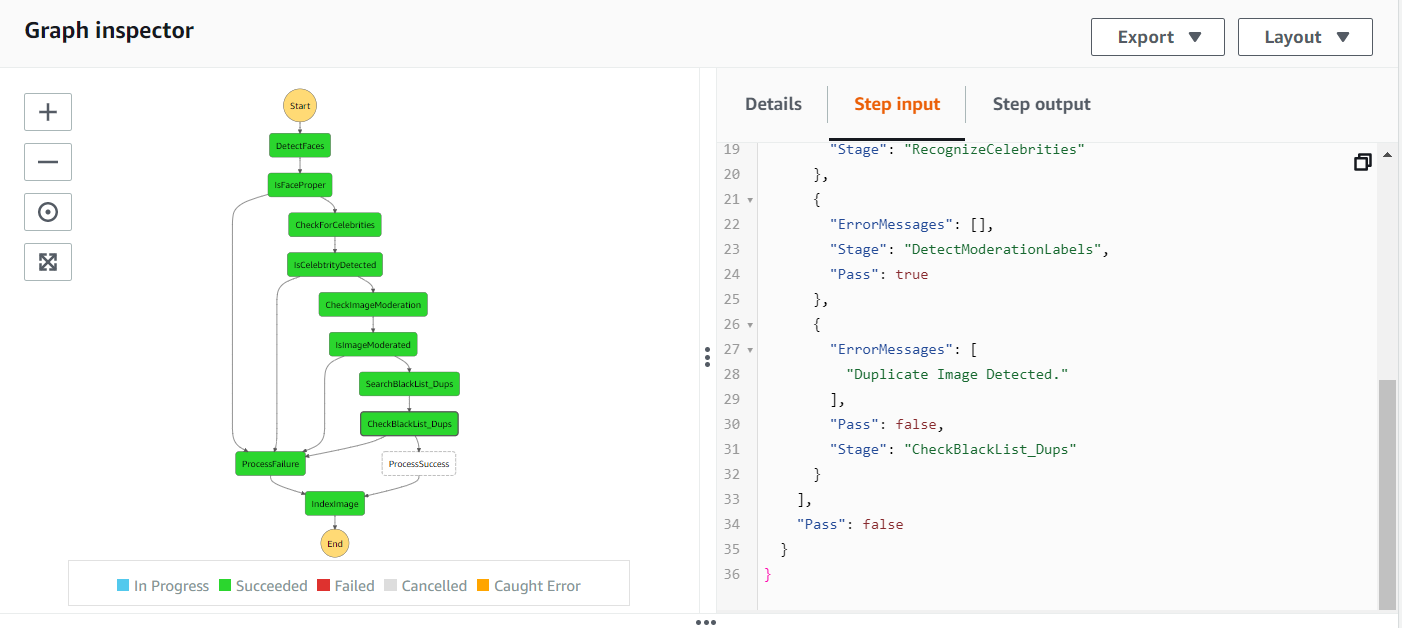
After Execution

This image passed all the constraints, and was successfully stored within the cluster through AWS Elasticsearch. The image will be pulled and referenced again when attempting to filter future images for duplicates.

1. **Duplicate Image Detection**



Input Image



The above image gave us an error: “A Duplicate Image was detected”.

The subject of this image is the same as used in test #7. As shown, even though the image uploaded was the subject without glasses, Rekognition was still able to compare with the previously successfully passed image and identify them as the same person shown in a previous image. Thus, returning as a duplicate image failure.

# References

Official AWS documentation: 12/10/2020 - Amazon

AWS CloudFormation user guide

-<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>

Working with Amazon S3 Buckets

- <https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html>

SearchFaces documentation

-<https://docs.aws.amazon.com/rekognition/latest/dg/API_SearchFaces.html>

DetectFaces documentation

-<https://docs.aws.amazon.com/rekognition/latest/dg/API_DetectFaces.html>

RecognizeCelebrities documentation

-<https://docs.aws.amazon.com/rekognition/latest/dg/API_RecognizeCelebrities.html>

AWS Lambda developer guide

- <https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html>

AWS Step Functions developer guide

- <https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html>

AWS Elasticsearch developer guide

-<https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/what-is-amazon-elasticsearch-service.html>

Amazon Rekognition developer guide

- <https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html>

# Glossary

**Bucket -** A public cloud storage resource utilized by AWS S3 which are similar to file folders, store objects, which consist of data and its descriptive metadata.

**Lambda function -** A process whose deployment and management is handled and run through AWS Lambda.

**Kibana -** An open source data visualization dashboard for Elasticsearch.

**Cluster -** In the context of AWS, a cluster is a grouping of containerized task or service instances.

**Container -** A container houses all of the needed resources for a program to run in a single package

**AWS (acronym) -** Amazon Web Services

**S3 (acronym) -** Simple Storage Service

**API (acronym) -** Application Programming Interface

**IAM (acronym) -** Identity and Access Management