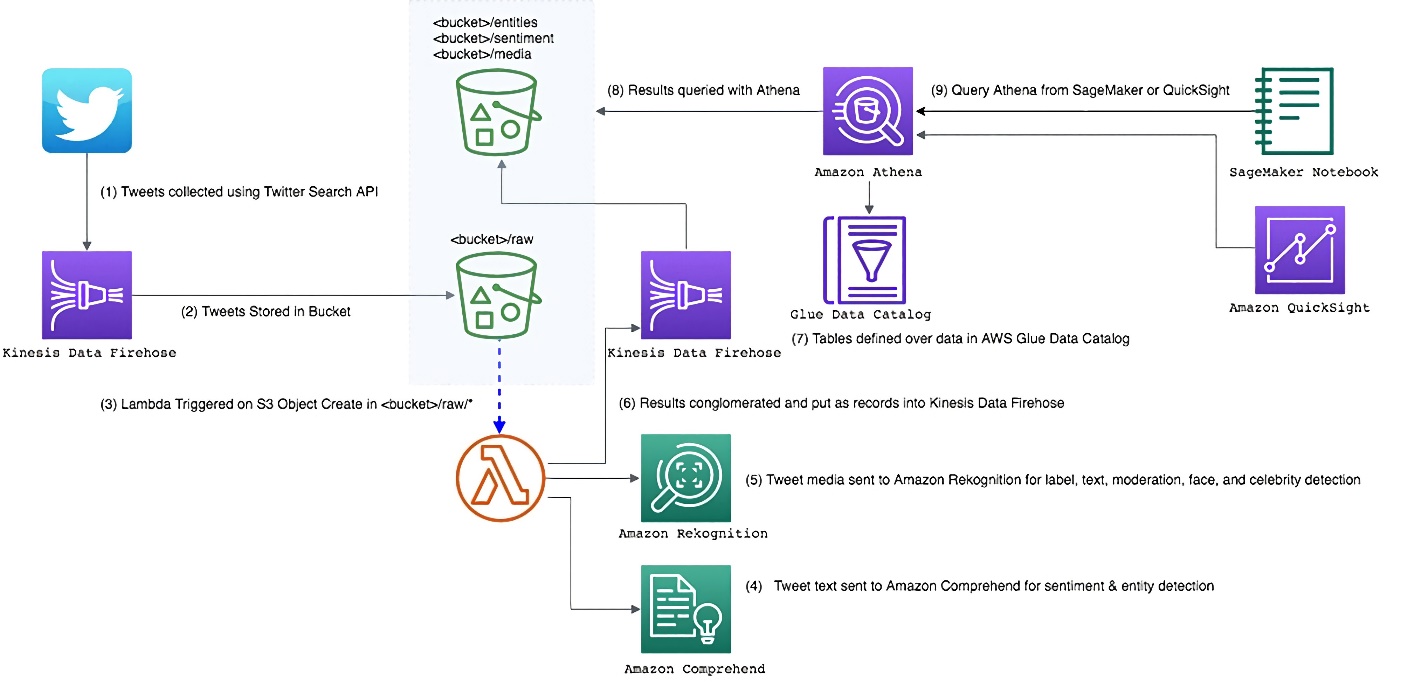
# AWS Exploring images on social media using Amazon Rekognition and Amazon Athena

If you’re like most companies, you wish to better understand your customers and your brand image. You’d like to track the success of your marketing campaigns, and the topics of interest—or frustration—for your customers. Social media promises to be a rich source of this kind of information, and many companies are beginning to collect, aggregate, and analyze the information from platforms like Twitter.

However, more and more social media conversations center around images and video; on one recent project, approximately 30% of all tweets collected included one or more images. These images contain relevant information that is not readily accessible without analysis.

## Overview of solution

The following diagram shows the solution components and how the images and extracted data flow through them.

These components are available through an [AWS CloudFormation](http://aws.amazon.com/cloudformation) template.

1. [Twitter Search API](https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html) collects Tweets.
2. [Amazon Kinesis Data Firehose](https://aws.amazon.com/kinesis/data-firehose/) dispatches the tweets to store in an [Amazon S3](http://aws.amazon.com/s3)
3. The creation of an S3 object in the designated bucket folder triggers a Lambda function.
4. The Lambda sends each tweet text to [Amazon Comprehend](https://aws.amazon.com/documentation/comprehend/) to detect sentiment (positive or negative), entity (real-world objects such as people, places, and commercial items), and precise references to measures such as dates and quantities. For more information, see [DetectSentiment](https://docs.aws.amazon.com/comprehend/latest/dg/API_DetectSentiment.html" \t "_blank) and [DetectEntity](https://docs.aws.amazon.com/comprehend/latest/dg/API_DetectEntities.html" \t "_blank) in the Amazon Comprehend Developer Guide.
5. The Lambda checks each tweet for media of type ‘photo’ in the tweet’s extended\_entities If the photo has either a JPG or PNG extension, the Lambda calls the following Amazon Rekognition APIs for each image:
   * Detect\_labels, to identify objects such as Person, Pedestrian, Vehicle, and Car in the image.
   * Detect\_moderation\_labels, to determine if an image or stored video contains unsafe content, such as explicit adult content or violent content.
   * If the detect\_labels API returns a Text label, detect\_text extracts lines, words, or letters found in the image.
   * If the detect\_labels API returns a Person label, the Lambda calls the following:
     + detect\_faces, to detect faces and analyze them for features such as sunglasses, beards, and mustaches.
     + recognize\_celebrities, to detect as many celebrities as possible in different settings, cosmetic makeup, and other conditions.

The results from all calls for a single image are combined into a single JSON record. For more information about these APIs, see [Actions](https://docs.aws.amazon.com/rekognition/latest/dg/API_Operations.html) in the Amazon Rekognition Developer Guide.

1. The results of the Lambda go to Kinesis Data Firehose. Kinesis Data Firehose batches the records and writes them to a designated S3 bucket and folder.
2. You can use [Amazon Athena](http://aws.amazon.com/athena) to build tables and views over the S3 datasets, then catalog these definitions in the [AWS Glue](https://aws.amazon.com/glue) Data Catalog. The table and view definitions make it much easier to query the complex JSON objects contained in these S3 datasets.
3. After the processed tweets land in S3, you can query the data with Athena.
4. You can also use [Amazon QuickSight](https://aws.amazon.com/quicksight) to visualize the data, or [Amazon SageMaker](https://aws.amazon.com/documentation/sagemaker/) or [Amazon EMR](http://aws.amazon.com/emr) to process the data further.

## Prerequisites

This walkthrough has the following prerequisites:

* An [AWS account](https://signin.aws.amazon.com/signin?redirect_uri=https%3A%2F%2Fportal.aws.amazon.com%2Fbilling%2Fsignup%2Fresume&client_id=signup).
* An app on Twitter. To create an app, see the [Apps section](https://developer.twitter.com/en/apps) of the Twitter Development website.
  + Create a consumer key (API key), consumer secret key (API secret), access token, and access token secret. The solution uses them as parameters in the AWS CloudFormation stack.

## Walkthrough

This process walks you through the following steps,

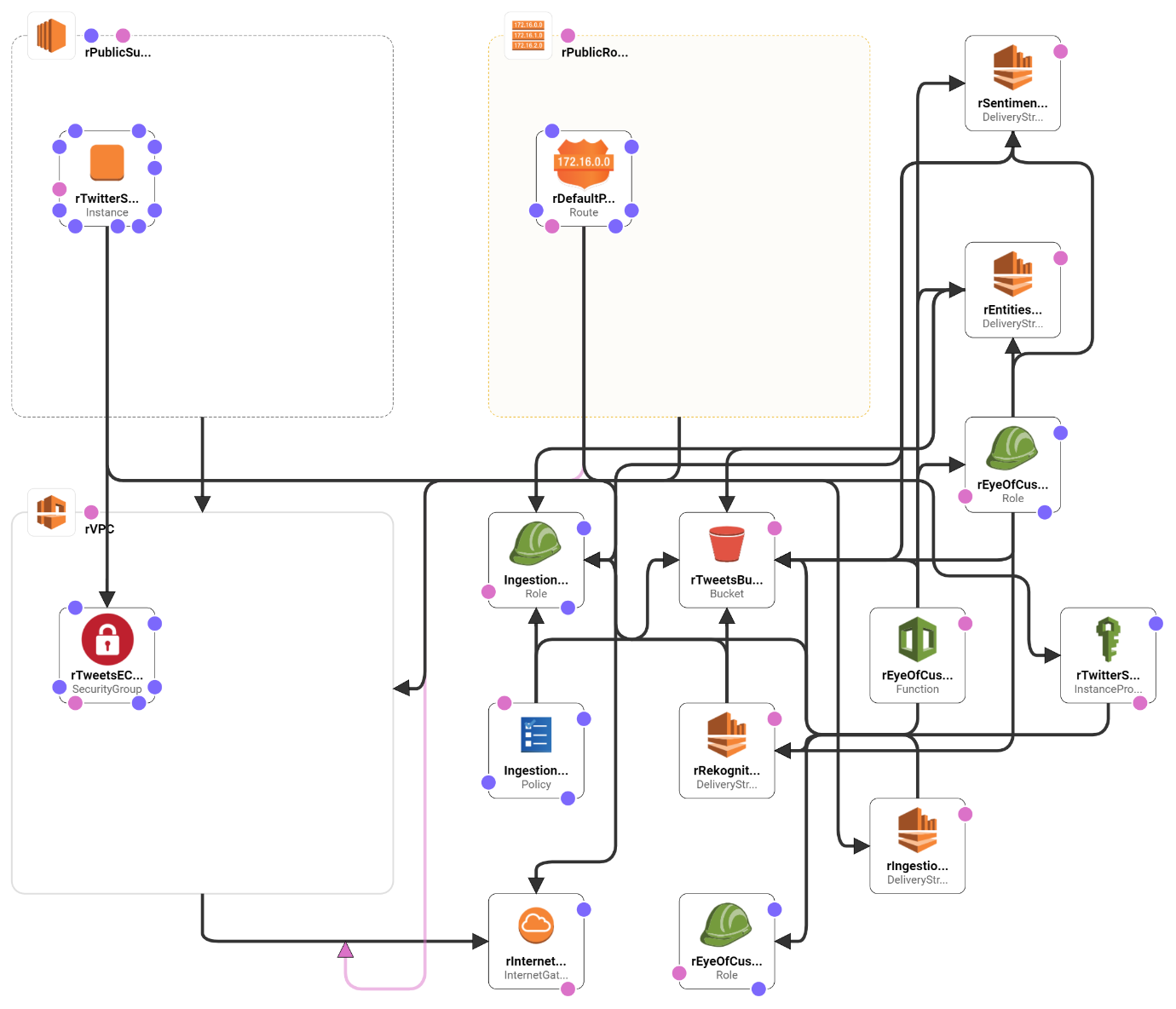
* Check the source code on Github <https://github.com/Ramstein/IncedoDataEngineeringProject> and its CloudFormation template.
* Launching the provided AWS [CloudFormation template](https://github.com/Ramstein/IncedoDataEngineeringProject/blob/master/IncedoProject/SMIAWRAofCustomer.yaml) and collecting tweets.
* Checking the stack-created datasets on S3.
* Creating views over the datasets using Athena.
* Exploring the data.

S3 stores the raw tweets and the Amazon Comprehend and Amazon Rekognition output in JSON format. You can use the Athena table and view definitions to flatten the complex JSON produced and extract your desired fields. This approach makes the data easier to access and understand.

## Launching the AWS CloudFormation template

This [AWS CloudFormation template is available on Github](https://github.com/Ramstein/IncedoDataEngineeringProject/blob/master/IncedoProject/SMIAWRAofCustomer.yaml) that creates all the ingestion components that appear in the previous diagram, except for the S3 notification for Lambda (the dotted blue line in the diagram).

1. In the AWS Management Console, upload the file template and launch the stack creation. This launches the AWS CloudFormation stack automatically into the us-east-1 Region.



1. After of above stack, for processing and analyzing the image data also from the social media posts, we can do them via somewhat a similar template using Amazon Rekogition and Athena. You can read the attachment provided named “AWS Exploring images on social media using Amazon Rekognition and Amazon Athena” in the project root dir to build that architecture. Follow the steps outlined, with the following changes:
   * Use the **Launch Stack** link from this post.
   * If the AWS Glue database socialanalyticsblog already exists (for example, if you completed the walkthrough from the previous post), change the name of the database when launching the AWS CloudFormation stack, and use the new database name for the rest of this solution.
   * For Twitter Languages, use ‘en’ (English) only. This post removed the Amazon Comprehend Translate capability for simplicity and to reduce cost.
   * Skip the section “Setting up S3 Notification – Call Amazon Translate/Comprehend from new Tweets.” This occurs automatically when launching the AWS CloudFormation stack by the “Add Trigger” Lambda function.
   * Stop at the section “Create the Athena Tables” and complete the following instructions in this post instead.

You can modify which terms to pull from the Twitter streaming API to be relevant to your company and your customers. This project considers the Banking and Financial Services (BFSI) related terms, around 50 considered in this project are…

Finance, Forex, Trading, Business, Currency, Globaltrade, Investment, investing, Stockmarket, Realestate, Markets, Economy, Fintech, Entrepreneur, SustainableFinace, SustanableInvesting, Blockchain, Crypto, Cryptocurrency, Airdrop, Bitcoin, Btc, Etherium, Ico, Altcoin, Cryptonews, 2020election, Trump2020, USElections, PresidentialElection, Brexit, BrexitDay, China, Tradewar, FinancialFreedom, PersonalFinance, FinancialPlanning, Mortgage, FinancialLiteracy, RetirementPlanning, DebtFree, FinancialAdvisor, Financialnews, Financialadvice, Financial

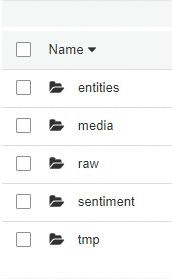
This implementation makes two Amazon Comprehend calls and up to five Amazon Rekognition calls per tweet. The cost of running this implementation is directly proportional to the number of tweets you collect. If you’d like to modify the terms to something that may retrieve tens or hundreds of tweets a second, for efficiency and cost management, consider performing batch calls or using AWS Glue with triggers to perform batch processing versus stream processing.

## Checking the S3 files

After the stack has been running for approximately five minutes, datasets start appearing in the S3 bucket (rTweetsBucket) that the AWS CloudFormation template created. Each dataset is represented as the following files sitting in a separate directory in S3:

* **Raw** – The raw tweets as received from Twitter.
* **Media** – The output from calling the Amazon Rekognition APIs.
* **Entities** – The results of Amazon Comprehend entity analysis.
* **Sentiment** – The results of Amazon Comprehend sentiment analysis.

See the following screenshot of the directory:



When you have enough data to explore (which depends on how popular your selected terms are and how frequently they have images), you can stop the Twitter stream producer, and stop or terminate the [Amazon EC2](http://aws.amazon.com/ec2) instance. This stops your charges from Amazon Comprehend, Amazon Rekognition, and EC2.

## Creating the Athena views

The next step is manually creating the Athena database and tables. For more information, see [Getting Started](https://docs.aws.amazon.com/athena/latest/ug/getting-started.html) in the Athena User Guide.

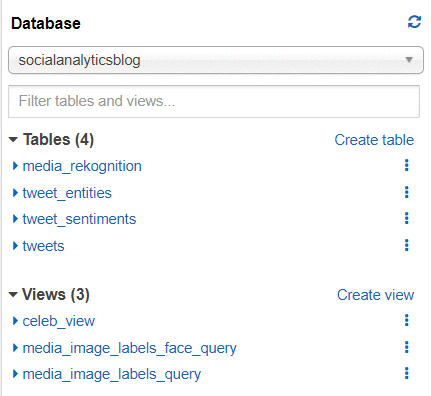
This is a great place to use [AWS Glue](https://aws.amazon.com/glue/) crawling features in your data lake architectures. The crawlers automatically discover the data format and data types of your different datasets that live in S3 (as well as relational databases and data warehouses). For more information, see [Defining Crawlers](https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html).

1. In the Athena console, in **Query Editor**, access the file [SQL](https://aws-bigdata-blog.s3.amazonaws.com/artifacts/EyeOfCustomer/EyeOfCustomer.sql). The AWS CloudFormation stack created the database and tables for you automatically.
2. Load the view create statements into the Athena query editor one by one, and execute. This step creates the views over the tables.

Compared to the prior post, the media\_rekognition table and the views are new. The tweets table has a new extended\_entities column for images and video metadata. The definitions of the other tables remain the same.

Your Athena database should look similar to the following screenshot. There are four tables, one for each of the datasets on S3. There are also three views, combining and exposing details from the media\_rekognition table:

* Celeb\_view focuses on the results of the recognize\_celebrities API
* Media\_image\_labels\_query focuses on the results from the detect\_labels API
* Media\_image\_labels\_face\_query focuses on the results from the detect\_faces API



Explore the table and view definitions. The JSON objects are complex, and these definitions show a variety of uses for querying nested objects and arrays with complex types. Now many of the queries can be relatively simple, thanks to the underlying table and view definitions encapsulating the complexity of the underlying JSON. For more information, see [Querying Arrays with Complex Types and Nested Structures](https://docs.aws.amazon.com/athena/latest/ug/rows-and-structs.html).

## Exploring the results

This section describes three use cases for this data and provides SQL to extract similar data. Because your search terms and timeframe are different from those in this post, your results will differ. This post used a set of Amazon-related terms. The tweet collector ran for approximately six weeks and collected approximately 9.5M tweets. From the tweets, there were approximately 0.5M photos, about 5% of the tweets. This number is low compared to some other sets of business-related search terms, where approximately 30% of tweets contained photos.

This post reviews four image use cases:

1. Buzz
2. Labels and faces
3. Suspect content
4. Exploring celebrities

### Buzz

Major topic areas represented by the links associated with the tweets often provide a good complement to the tweet language content topics surfaced via natural language processing. For more information, see [Build a social media dashboard using machine learning and BI services](https://aws.amazon.com/blogs/machine-learning/build-a-social-media-dashboard-using-machine-learning-and-bi-services/).

The first query is which websites the tweets are linked to. The following code shows the top domain names linked from the tweets:

SELECT lower(url\_extract\_host(URL.expanded\_url)) AS domain,

count(\*) AS count

FROM

(SELECT \*

FROM "tweets"

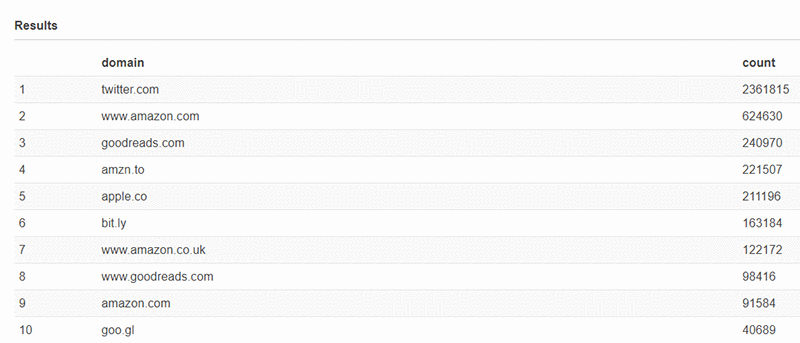
CROSS JOIN UNNEST (entities.urls) t (url))

GROUP BY 1

ORDER BY 2 DESC

LIMIT 10;

The following screenshot shows the top 10 domains returned:



Links to Amazon websites are frequent, and several different properties are named, such as [amazon.com](http://amazon.com/), [amazon.co.uk](http://www.amazon.co.uk/), and goodreads.com.

Further exploration shows that many of these links are to product pages on the Amazon website. It’s easy to recognize these links because they have /dp/ (for detail page) in the link. You can get a list of those links, the images they contain, and the first line of text in the image (if there is any), with the following query:

SELECT tweetid,

user\_name,

media\_url,

element\_at(textdetections,1).detectedtext AS first\_line,

expanded\_url,

tweet\_urls."text"

FROM

(SELECT id,

user.name AS user\_name,

text,

entities,

url.expanded\_url as expanded\_url

FROM tweets

CROSS JOIN UNNEST (entities.urls) t (url)) tweet\_urls

JOIN

(SELECT media\_url,

tweetid,

image\_labels.textdetections AS textdetections

FROM media\_rekognition) rk

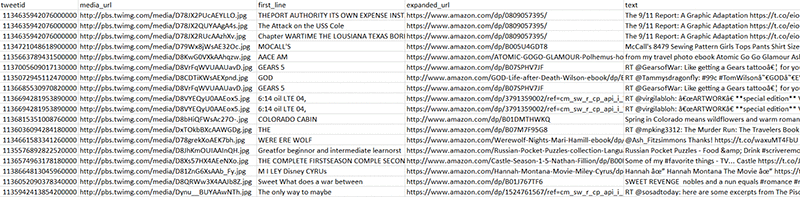
ON rk.tweetid = tweet\_urls.id

WHERE lower(url\_extract\_host(expanded\_url)) IN ('www.amazon.com', 'amazon.com', 'www.amazon.com.uk', 'amzn.to')

AND NOT position('/dp/' IN url\_extract\_path(expanded\_url)) = 0 -- url links to a product

LIMIT 50;

The following screenshot shows some of the records returned by this query. The first\_line column shows the results returned by the detect\_text API for the image URL in the media\_url column.



Many of the images do contain text. You can also identify the products the tweet linked to; many of the tweets are product advertisements by sellers, using images that relate directly to their product.

### Labels and faces

You can also get a sense of the visual content of the images by looking at the results of calling the Amazon Rekognition detect\_labels API. The following query finds the most common objects found in the photos:

SELECT label\_name,

COUNT(\*) AS count

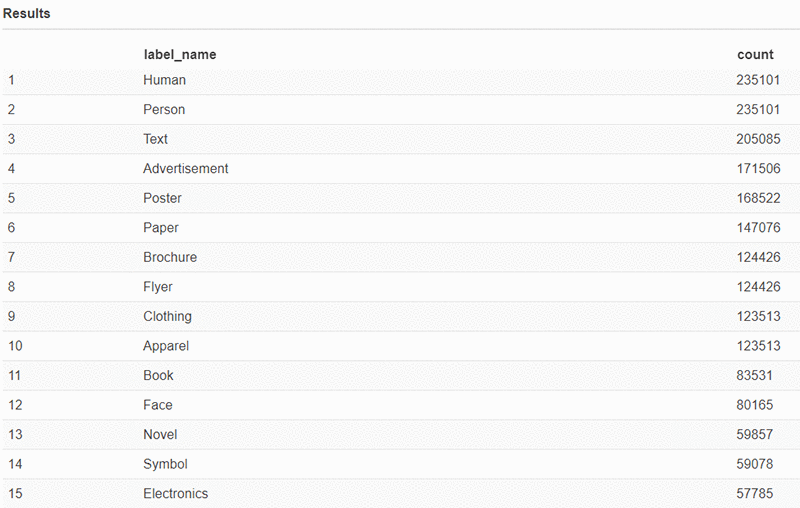
FROM media\_image\_labels\_query

GROUP BY label\_name

ORDER BY COUNT(\*) desc

LIMIT 50;

The following screenshot shows the results of that request. The most popular label by far is Human or Person, with Text, Advertisement, and Poster coming soon after. The novel is further down the list. This result reflects the most popular product being tweeted about on the Amazon website—books.



You can explore the faces further by looking at the results of the detect\_faces API. That API returns details for each face in the image, including the gender, age range, face position, whether the person is wearing sunglasses or has a mustache, and the expression(s) on their face. Each of these features also has a confidence level associated with it. For more information, see [DetectFaces](https://docs.aws.amazon.com/rekognition/latest/dg/API_DetectFaces.html" \t "_blank) in the Amazon Rekognition Developer Guide.

The view media\_image\_labels\_face\_query unnests many of these features from the complex JSON object returned by the API call, making the fields easy to access.

You can explore the view definition for media\_image\_labels\_face\_query, including the use of the [reduce operator](https://prestodb.github.io/docs/current/functions/array) on the array of (emotion,confidence) pairs that Amazon Rekognition returned to identify and return the expression category with the highest confidence score associated with it, and associate the name top\_emotion with it. See the following code:

reduce(facedetails.emotions, element\_at(facedetails.emotions, 1), (s\_emotion, emotion) -> IF((emotion.confidence > s\_emotion.confidence), emotion, s\_emotion), (s) -> s) top\_emotion

You can then use the exposed field, top\_emotion. See the following code:

SELECT top\_emotion.type AS emotion ,

top\_emotion.confidence AS emotion\_confidence ,

milfq.\* ,

"user".id AS user\_id ,

"user".screen\_name ,

"user".name AS user\_name ,

url.expanded\_url AS url

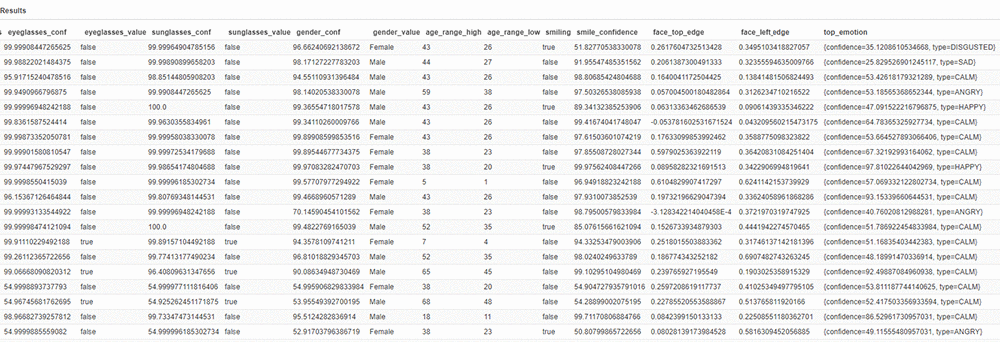
FROM media\_image\_labels\_face\_query milfq

INNER JOIN tweets

ON tweets.id = tweetid, UNNEST(entities.urls) t (url)

WHERE position('.amazon.' IN url.expanded\_url) > 0;

The following screenshot shows columns from the middle of this extensive query, including glasses, age range, and where the edges of this face are positioned. This last detail is useful when multiple faces are present in a single image, to distinguish between the faces.



You can look at the top expressions found on these faces with the following code:

SELECT top\_emotion.type AS emotion,

COUNT(\*) AS "count"

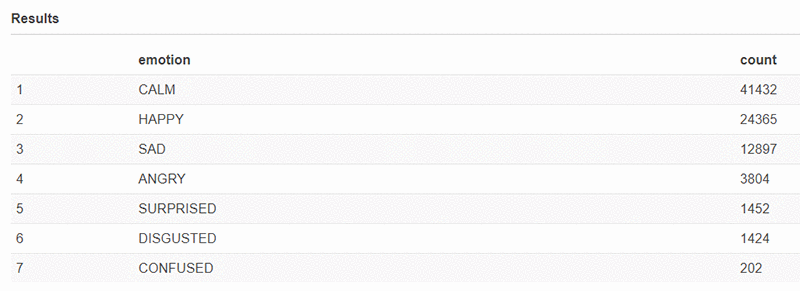
FROM media\_image\_labels\_face\_query milfq

WHERE top\_emotion.confidence > 50

GROUP BY top\_emotion.type

ORDER BY 2 desc;

The following screenshot of the query results shows that CALM is the clear winner, followed by HAPPY. Oddly, there are far fewer confused than disgusted expressions.



### Suspect content

A topic of frequent concern is whether there is content in the tweets or the associated images, that should be moderated. One of the Amazon Rekognition APIs called by the Lambda for each image is moderation\_labels, which returns labels denoting the category of the content found, if any. For more information, see [Detecting Unsafe Content](https://docs.aws.amazon.com/rekognition/latest/dg/moderation.html).

The following code finds tweets with suspect images. Twitter also provides a possibly\_sensitive flag based solely on the tweet text.

SELECT tweetid,

possibly\_sensitive,

transform(image\_labels.moderationlabels, ml -> ml.name) AS moderationlabels,

"mediaid", "media\_url" ,

tweets.text,

"url"."expanded\_url" AS url ,

(CASE WHEN ("substr"("tweets"."text", 1, 2) = 'RT') THEN

true

ELSE false END) "isretweet"

FROM media\_rekognition

INNER JOIN tweets

ON ("tweets"."id" = "tweetid"), UNNEST("entities"."urls") t (url)

WHERE cardinality(image\_labels.moderationlabels) > 0

OR possibly\_sensitive = True;

The following screenshot shows the first few results. For many of these entries, the tweet text or the image may contain sensitive content, but not necessarily both. Including both criteria provides additional safety.



Note the use of the [transform](https://prestodb.github.io/docs/current/functions/array) construct in the preceding query to map over the JSON array of moderation labels that Amazon Rekognition returned. This construct lets you transform the original content of the moderation labels object (in the following array) into a list containing only the name field:

[{confidence=52.257442474365234, name=Revealing Clothes, parentname=Suggestive}, {confidence=52.257442474365234, name=Suggestive, parentname=}]

You can filter this query to focus on specific types of unsafe content by filtering on specific moderation labels. For more information, see [Detecting Unsafe Content](https://docs.aws.amazon.com/rekognition/latest/dg/moderation.html).

A lot of these tweets have product links embedded in the URL. URLs for the Amazon.com website have a pattern to them: any URL with /dp/ in it is a link to a product page. You could use that to identify the products that may have explicit content associated with them.

### Exploring celebrities

One of the Amazon Rekognition APIs that Lambda called for each image was recognize\_celebrity. For more information, see [Recognizing Celebrities in an Image](https://docs.aws.amazon.com/rekognition/latest/dg/celebrities-procedure-image.html).

The following code helps determine which celebrities appear most frequently in the dataset:

SELECT name as a celebrity,

COUNT (\*) as count

FROM celeb\_view

GROUP BY name

ORDER BY COUNT (\*) desc;

The result counts instances of celebrity recognitions and counts an image with multiple celebrities multiple times.

For example, assume there is a celebrity with the label JohnDoe. To explore their images further, use the following query. This query finds the images associated with tweets in which JohnDoe appeared in the text or the image.

SELECT cv.media\_url,

COUNT (\*) AS count ,

detectedtext

FROM celeb\_view cv

LEFT JOIN -- left join to catch cases with no text

(SELECT tweetid,

mediaid,

textdetection.detectedtext AS detectedtext

FROM media\_rekognition , UNNEST(image\_labels.textdetections) t (textdetection)

WHERE (textdetection.type = 'LINE'

AND textdetection.id = 0) -- get the first line of text

) mr

ON ( cv.mediaid = mr.mediaid

AND cv.tweetid = mr.tweetid )

WHERE ( ( NOT position('johndoe' IN lower(tweettext)) = 0 ) -- JohnDoe IN text

OR ( (NOT position('johndoe' IN lower(name)) = 0) -- JohnDoe IN image

AND matchconfidence > 75) ) -- with pretty good confidence

GROUP BY cv.media\_url, detectedtext

ORDER BY COUNT(\*) DESC;

The recognize\_celebrity API matches each image to the closest-appearing celebrity. It returns that celebrity’s name and related information, along with a confidence score. At times, the result can be misleading; for example, if a face is turned away, or when the person is wearing sunglasses, it can be difficult to identify correctly. In other instances, the API may choose an image model because of its similarity to a celebrity. It may be beneficial to combine this query with logic using the face\_details response, to check for glasses or face position.

## Cleaning up

To avoid incurring future charges, delete the AWS CloudFormation stack, and the contents of the S3 bucket created.

## Conclusion

This post showed how to start exploring what your customers are saying about you on social media using images. The queries in this post are just the beginning of what’s possible. To better understand the totality of the conversations your customers are having, you can combine the capabilities from this post with the results of running natural language processing against the tweets.

This entire processing, analytics, and machine learning pipeline—starting with Kinesis Data Firehose, using Amazon Comprehend to perform sentiment analysis, Amazon Rekognition to analyze photographs, and Athena to query the data—is possible without spinning up any servers.

[Amazon Athena](https://aws.amazon.com/blogs/machine-learning/category/analytics/amazon-athena/), [Amazon Rekognition](https://aws.amazon.com/blogs/machine-learning/category/artificial-intelligence/amazon-rekognition/), [Analytics](https://aws.amazon.com/blogs/machine-learning/category/analytics/), [Artificial Intelligence](https://aws.amazon.com/blogs/machine-learning/category/artificial-intelligence/)