**Build a Log analyser for service desk tickets using machine learning and BI services**

**Created by: HEMANTH KUMAR VADUPU**

**Date: 10/NOV/2019**

In ths project, my goal is to Building a Log analyser for service desk tickets based on resolution knowledge base match and retrieve to demonstrate automated resolution. Additional innovations can be achieved based on using AWS lambda functions and Alexa integrations in terms of resolution methods

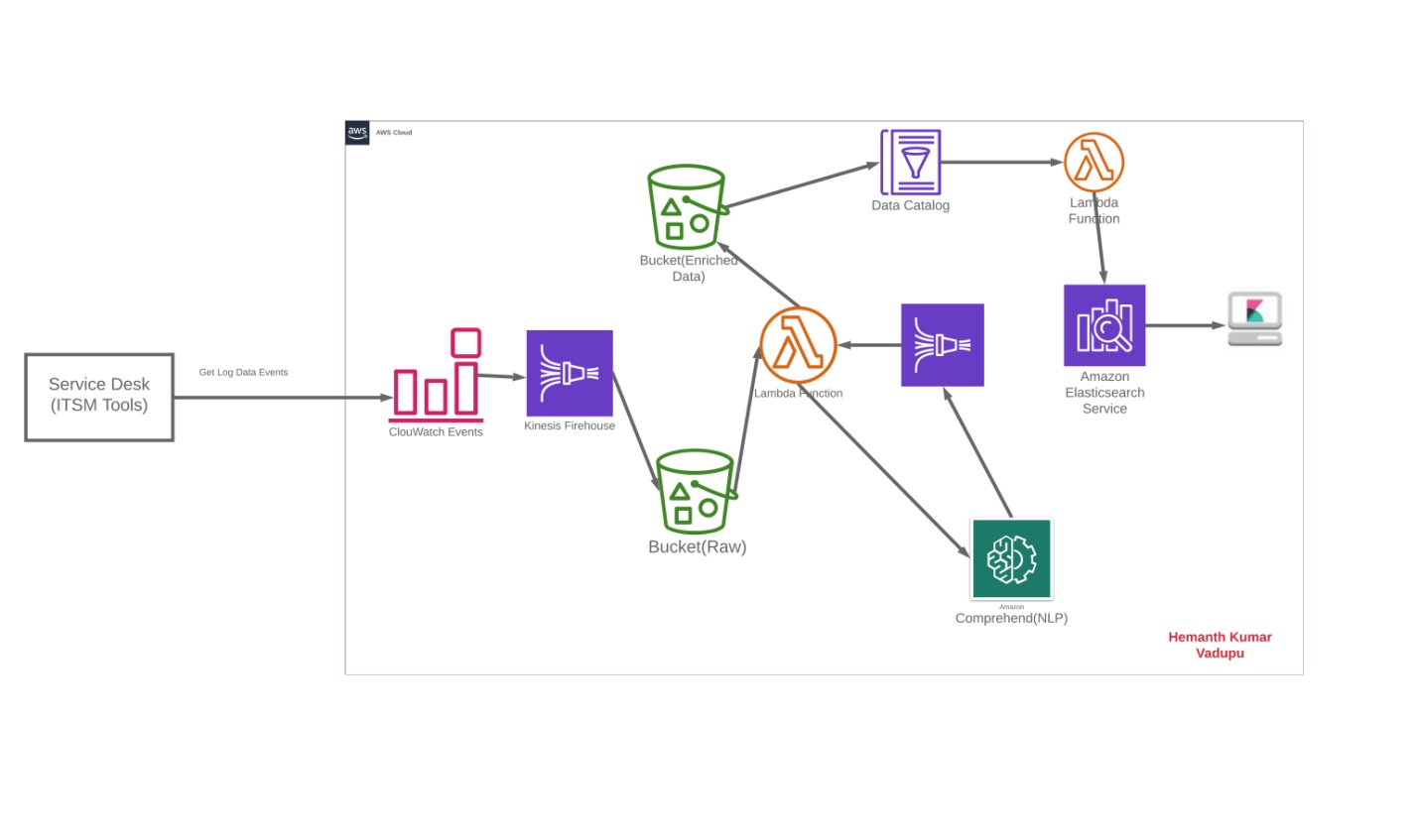
Ability to process and analyze data in real-time is essential to do things such as continuously monitor applications hosted on AWS and to have service assurance with zero downtime.

We’ll leverage API-driven ML services that allow developers to easily add intelligence to any application, such as computer vision, speech, language analysis, and chatbot functionality simply by calling a highly available, scalable, and secure endpoint. These building blocks will be put together with very little code, by leveraging serverless offerings within AWS.

**Prerequisites**

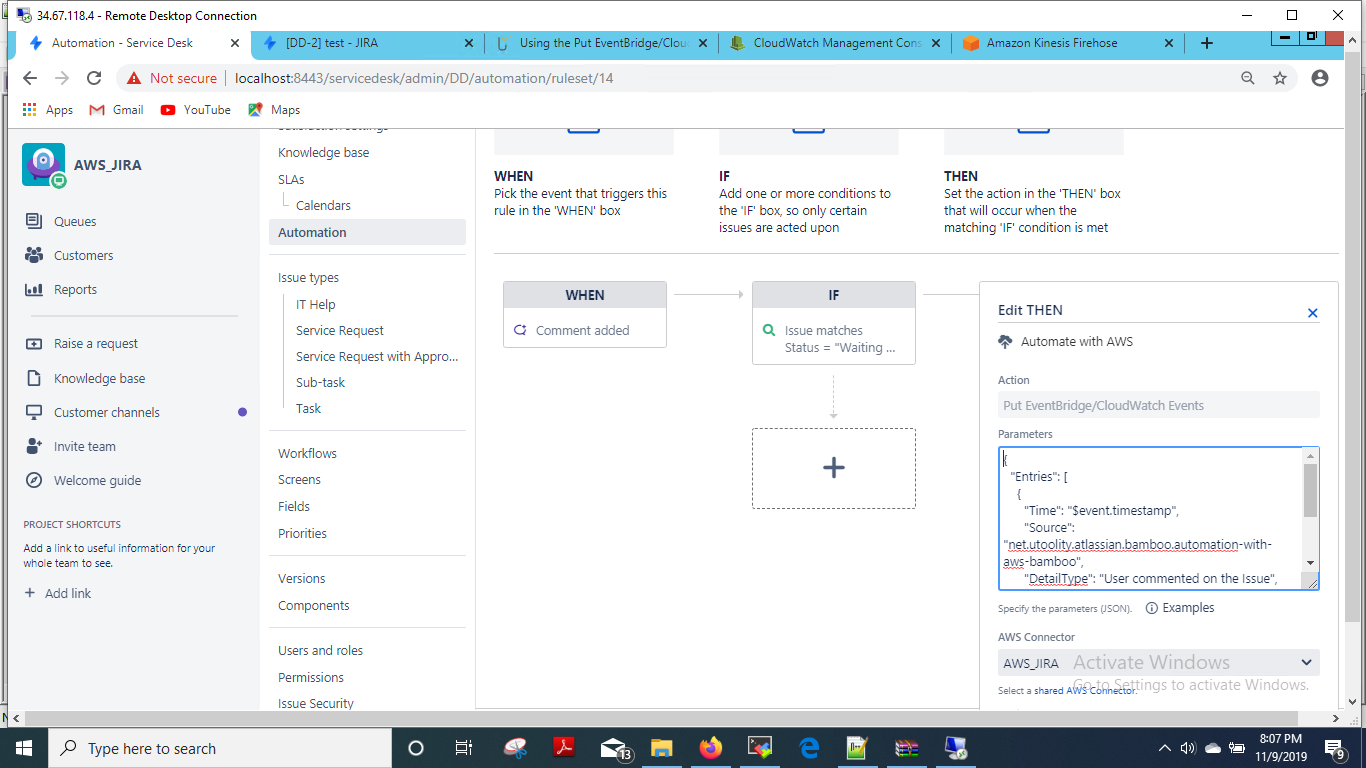
## Here are the Prerequisites for the Solution:

* Leverage Amazon Kinesis Data Firehose to easily capture, prepare, and load real-time data streams into data stores, data warehouses, and data lakes. In this example, we’ll use Amazon S3.
* Data Lake is like a large container which is very similar to real lake and rivers. Just like in a lake you have multiple tributaries coming in, a data lake has structured data, unstructured data, machine to machine, logs flowing through in real-time.
* Trigger AWS Lambda to analyze the tickets using Amazon Translate and Amazon Comprehend, two fully managed services from AWS. With only a few lines of code, these services will allow us to translate between languages and perform natural language processing (NLP) on the tweets.
* Leverage separate Kinesis data delivery streams within Amazon Kinesis Data Firehose to write the analyzed data back to the data lake.
* Leverage ElasticSearch makes it easy to index data in different schemas and make it available for users to explore.
* Build a set of dashboards using Kibana.
* ITSM tool (Atlassian Service Desk) for IT incident management.

**Architecture Diagram:**

**FIRST: Collect the Data from Service desk:**

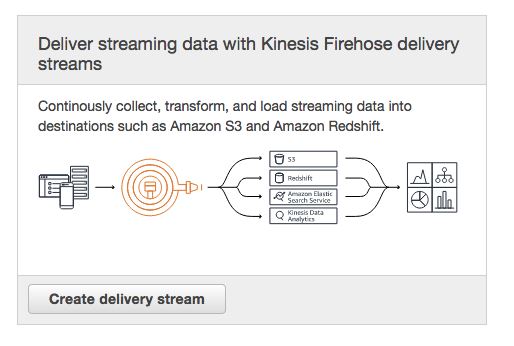
User submit incident tickets to IT Service Management Tools (ITSM). Tickets can be user generated or machine generated. With Service Desk automation rules to notify and automate all the things with AWS Cloud Watch Events.

****

**SECOND: Creating a kinesis Firehouse:**

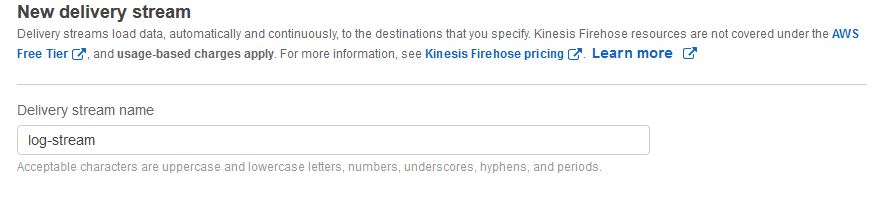
In this task, we will create a Kinesis Firehose delivery stream. It will transform incoming s3. Amazon S3 provides low cost, highly durable object storage that can store any form or format of data.

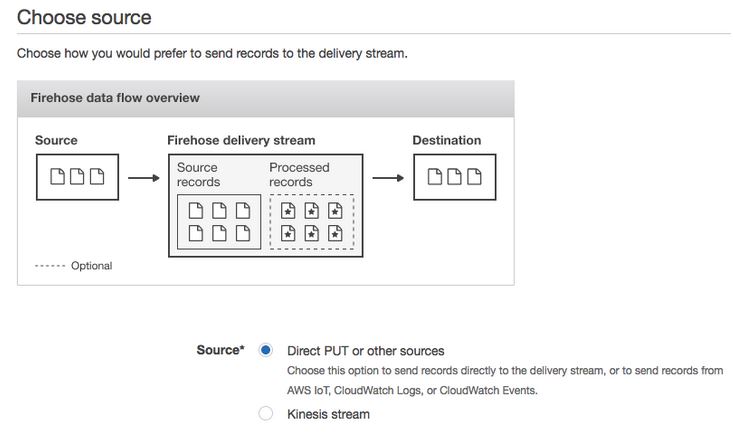
On the **Kinesis** service, click "Get started" and "Create delivery stream".

****

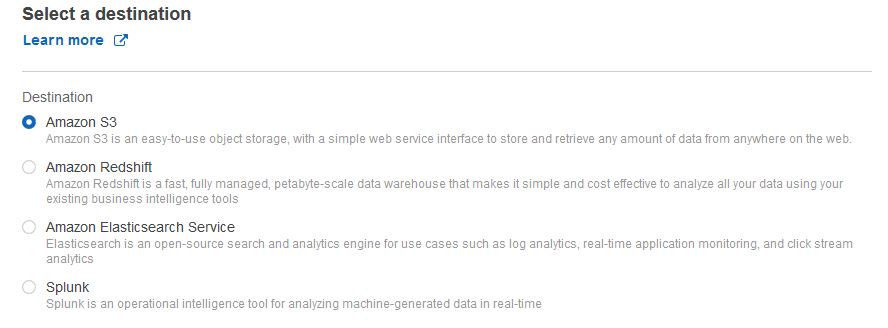
For **Delivery stream name**, enter "log-stream".

The information on the screen explains options for accepting incoming streaming data.

****

****

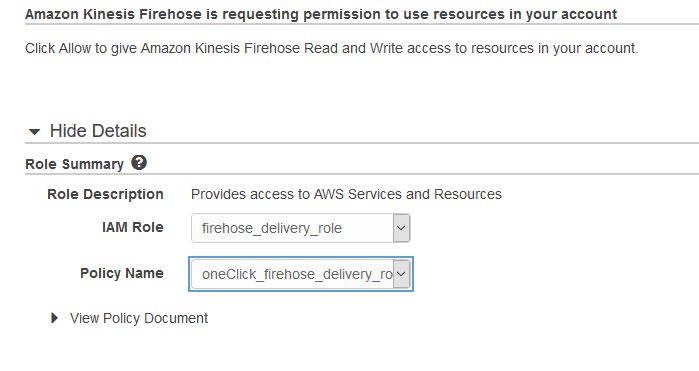
Click on “next” and select the destination Bucket.



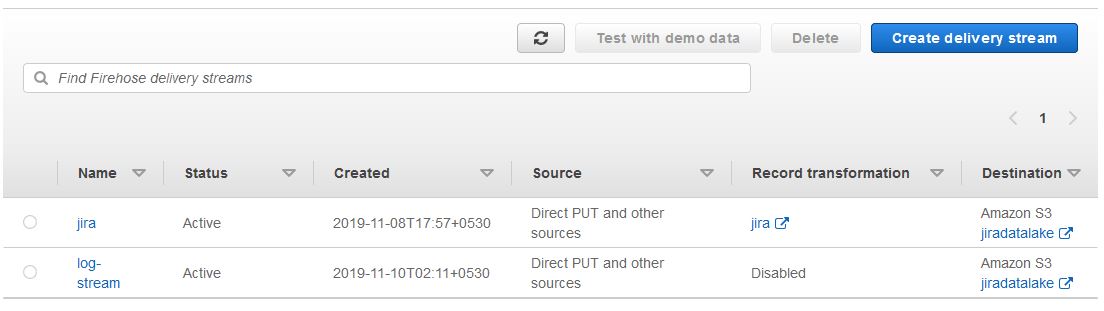
Choose existing bucket or create new bucket



Create new role or choose existing role for Kinesis Firehouse



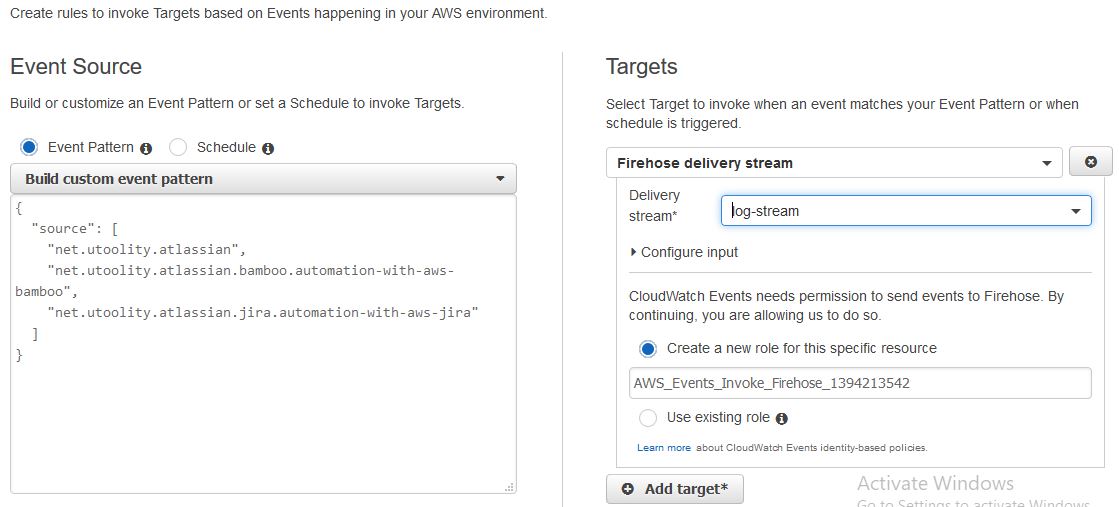
Click on “Allow” and click on Create delivery stream.



**THIRD: Create rule cloudwatch Events:**

As per Atlassian Service Desk logs data can be collected from the Cloud Watch events

Build or customize an Event Pattern or set a Schedule to invoke Targets.



Customize Evenent Pattern preview for jira:

**{**

**"source": [**

**"net.utoolity.atlassian",**

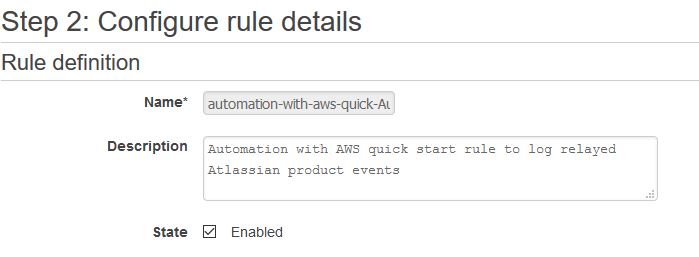
**"net.utoolity.atlassian.bamboo.automation-with-aws-bamboo",**

**"net.utoolity.atlassian.jira.automation-with-aws-jira"**

**]**

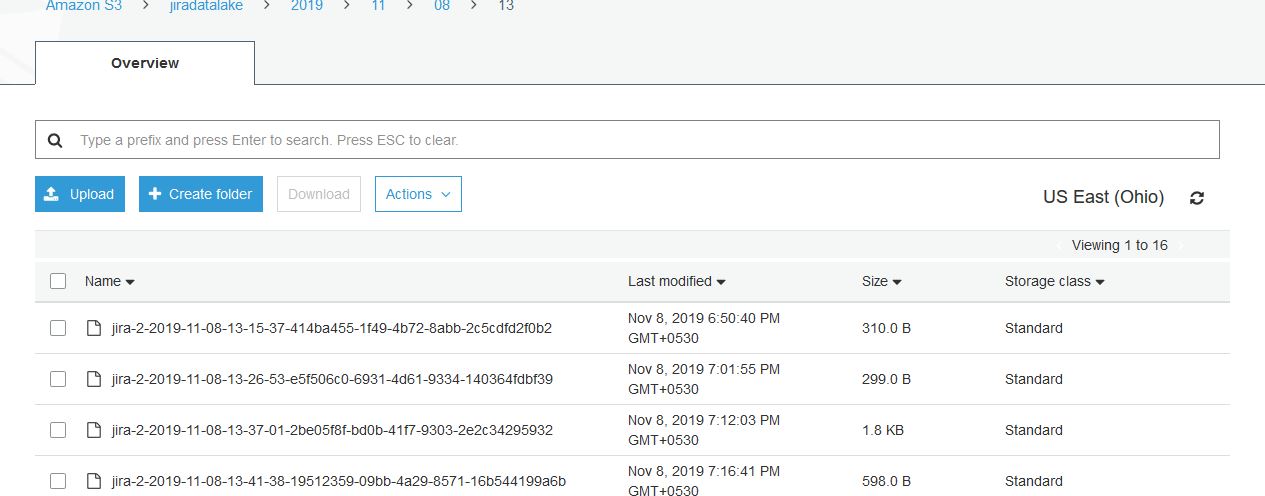
**}**

Configure rule and Create the cloudwatch rule.



**NOTE:** Whenever the cloudwatch event collected the log data frmm the service desk , then invoke the kinesis firhouse stream delivey , then the firehouse sent streaming data to the Destinaton S3 Bucket.

The firehouse streaming data stored in the S3 Bucket as,



### Setting up S3 Notification – Call Comprehend from Service desk logs:

The Lambda function calls Amazon Translate and Amazon Comprehend to perform language translation and natural language processing (NLP) on logs. The function uses Amazon Kinesis to write the analyzed data to Amazon S3.

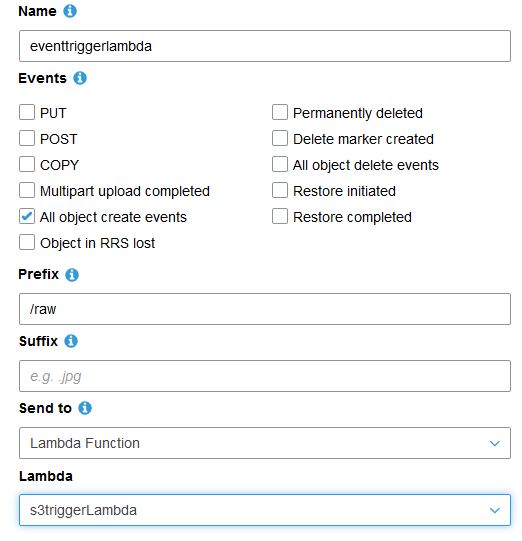
The Amazon Comprehend custom classification API was good choice since it is built ground-up for text classification. With Amazon Comprehend, we didn’t have to pick an algorithm, tune it and re-train our model looking for the highest accuracy – the API did this automatically.

we will have you add the S3 notification so that the Lambda function is invoked when new tweets are written to S3:

1. Under Add Triggers, select the S3 trigger.
2. Then configure the trigger with the new S3 bucket that created with the ‘raw/’ prefix. The event type should be Object Created (All).

Following least privilege patterns, the IAM role that the Lambda function has been assigned only has access to the S3 bucket

The following diagram shows an example:



## Fourth: Setting up a Lambda function for sentiment analysis

## There are multiple natural language and text processing frameworks or services available to use with Lambda, including but not limited to Amazon Comprehend, TextBlob, Pattern, and NLTK. Pick one based on the nature of your system:  the type of interaction, languages supported, and so on. For this post, I picked Amazon Comprehend, which uses natural language processing (NLP) to extract insights and relationships in text.

## The Lambda function is straightforward. It analyses the input transcript field of the Amazon Lex event. Based on the overall sentiment value, it generates a response message with next step instructions. When the sentiment is neutral, positive, or mixed, the response leaves it to Amazon Lex to decide what the next steps should be. It adds to the response overall sentiment value as an additional session attribute, along with slots’ values received as an input.

For this walkthrough, you create a Lambda function using the AWS Management Console:

1. Open the Lambda console.
2. Choose **Create Function**.
3. Choose **Author from scratch** (no blueprint).
4. For **Runtime**, choose Python 3.6.
5. For **Role**, choose **Create a custom role**. The custom execution role allows the function to detect sentiments, create a log group, stream log events, and store the log events.
6. Enter the following values:
   * For **Role Description**, enter *Lambda execution role permissions*.
   * For **IAM Role**, choose **Create an IAM role**.
   * For **Role Name**, enter AmazonComprehencedservicerole
   * For **Policy**, use the following policy:

JSON:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "arn:aws:logs:\*:\*:\*"

},

{

"Action": [

"comprehend:DetectDominantLanguage",

"comprehend:DetectSentiment"

],

"Effect": "Allow",

"Resource": "\*"

}

]

}

* 1. Choose **Create function**.
  2. Copy/paste the following code to the editor window.
  3. Click on the Save in the top corner of the window.

import boto3

**import json**

**import boto3**

**comprehend = boto3.client(service\_name='comprehend', region\_name='region')**

**text = "It is raining today in Seattle"**

**print('Calling DetectSentiment')**

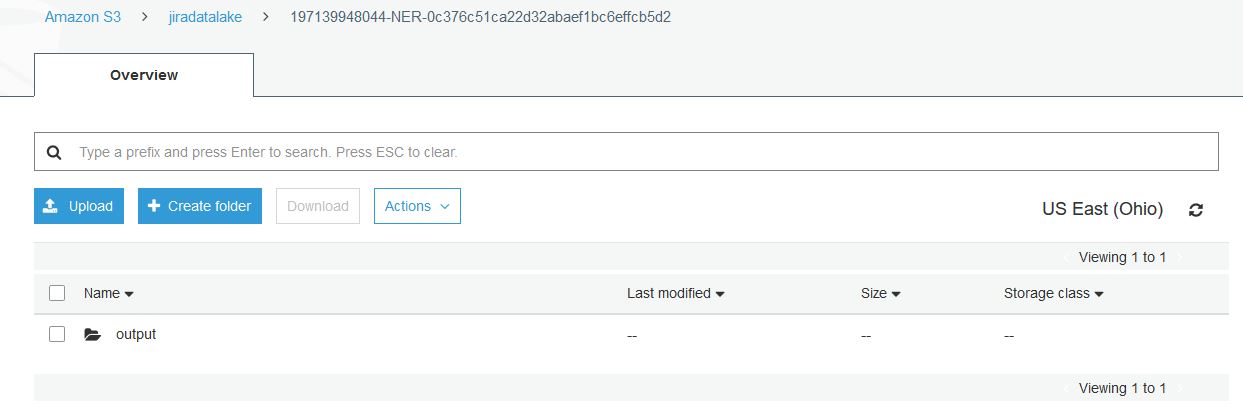
**print(json.dumps(comprehend.detect\_sentiment(Text=text, LanguageCode='en'), sort\_keys=True, indent=4))**

**print('End of DetectSentiment\n')**

Output of the lambda function as shown in the figure,



The function uses Amazon Kinesis to write the analyzed data to Amazon S3 /output.

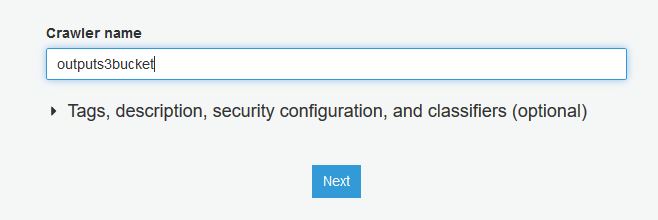


**FIFTH: Setting up Glue for central repository for your metadata**

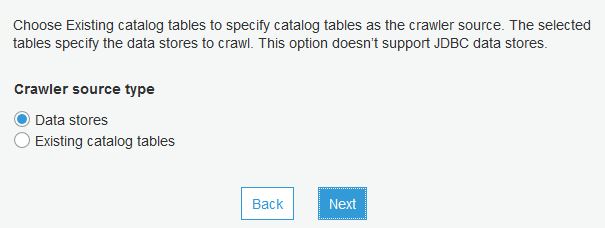
Assuming that most of the data come in its raw form, we now need a way to extract meaning from it. Let’s start by attempting to interpolate the schema from the data that we have at our disposal. While AWS Glue can be used to run ETL jobs and store metadata, it also has the ability to catalog it, ie. Interpolate the schema of the data by crawl through it using crawlers. AWS Glue crawlers are able to interpret many of the popular formats, such as JSON and Avro, and if you need, you can develop your own crawler to categorize your data. Extracted schema data is now stored in the AWS Glue Data catalog, and can be freely accessed.

This is a great place to leverage AWS Glue crawling features in your data lake architectures. The crawlers will automatically discover the data format and data types of your different datasets that live in Amazon S3.

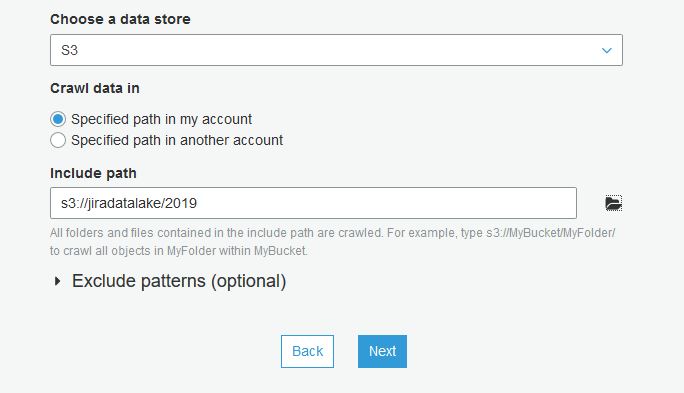
1. I n the AWS Management Console, navigate to AWS Glue. In the left navigation pane under **Data Catalog**, choose **Crawlers and Click on Add Crawlers.**
2. Enter Crawler Name and click on next.

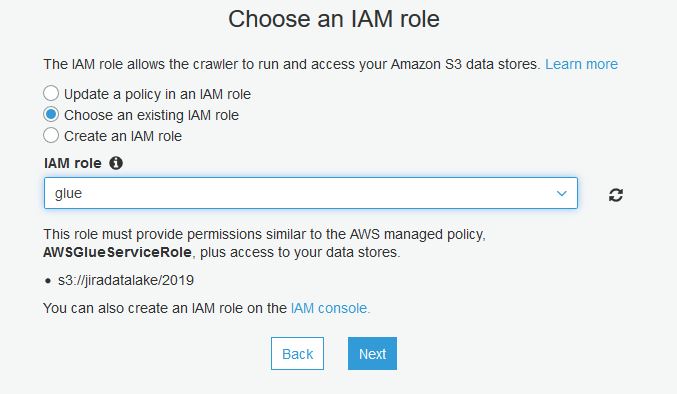


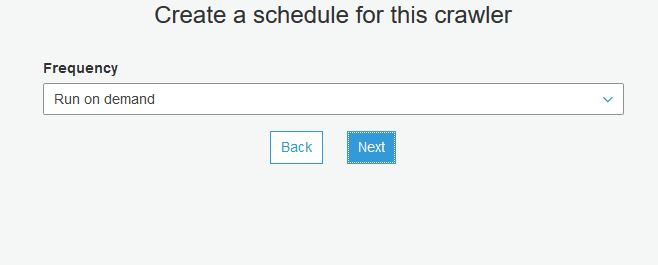
1. **Specify the Crawler source type and click on next.**



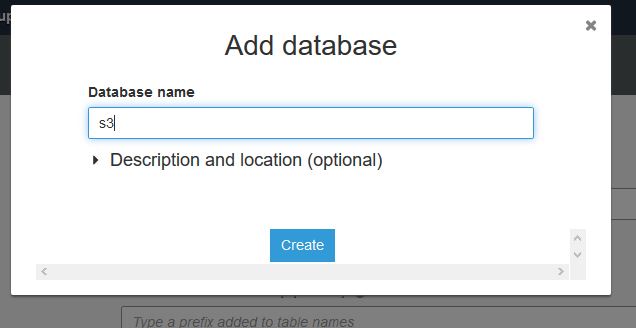
1. **Choose data store as S3, and Enter the path of the data store in s3.**



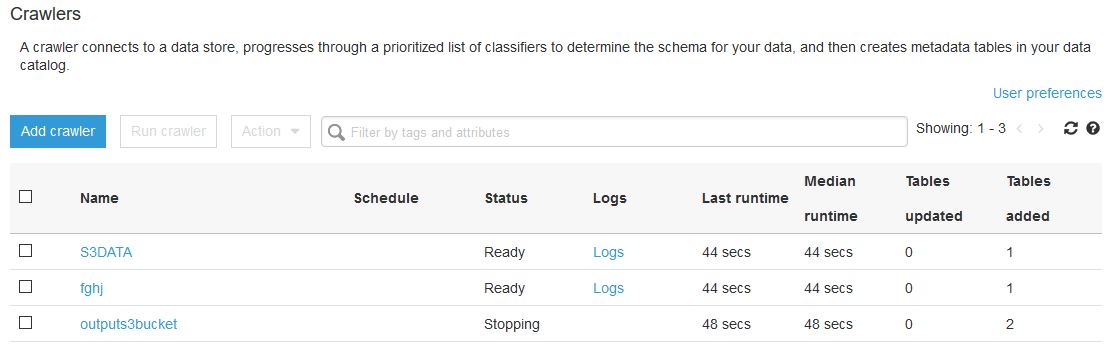
1. **Create IAM role or choose from the existing role.**
2. **Schedule the crawler as On-demand and click on next.**



1. Add database name , click on create and then enter.

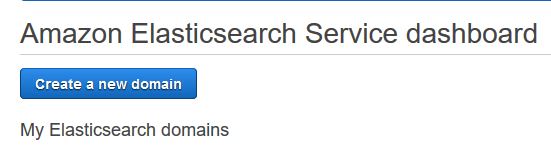


1. Review all steps and create cawler.

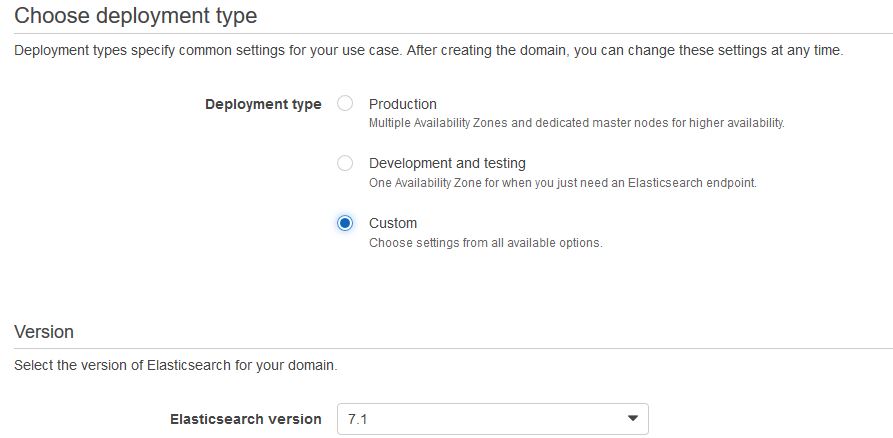


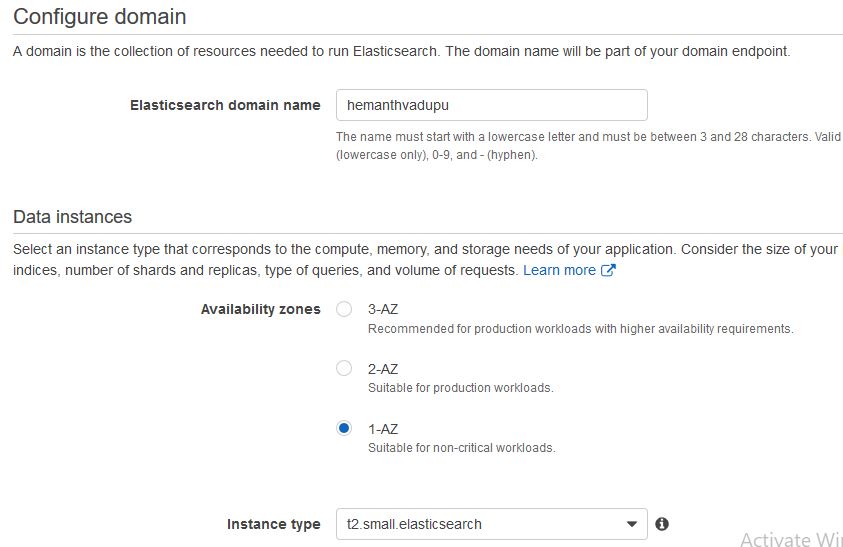
**SIXTH: Create Elasticdsearch for domain**

1. Navigate to elastic search domain and click on create new domain.

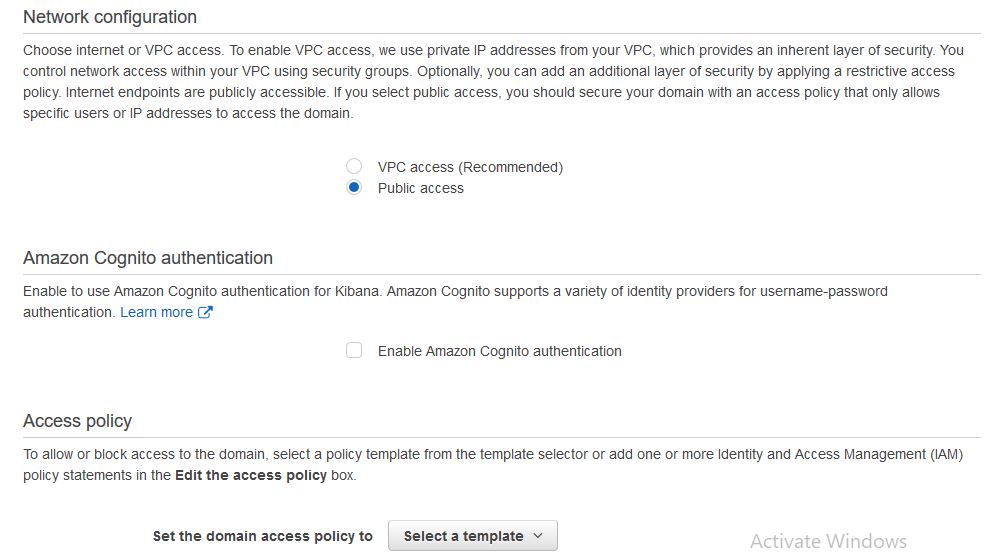


2.Configure the Cluster details as,

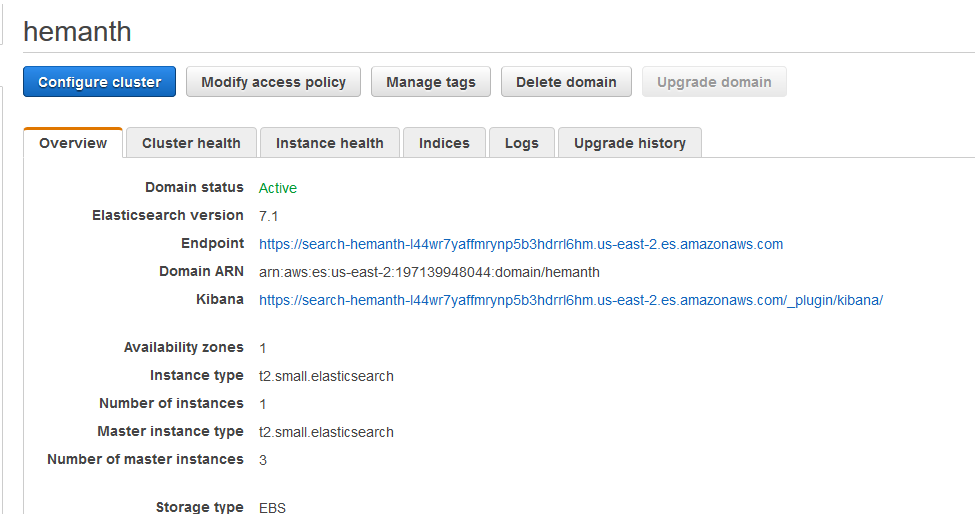




1. Configure the Network detail and create elastic search role.



1. Click on create.



The endpoint url of elastic cluster redirected to Kibana dashboard.

Now that you’ve made the first step towards better understanding the ingested data, let’s add additional information so that you can better manage and analyze your data. How do we do that? This is where the Lambda comes into the fold.

**import boto3** # aws sdk

**from elasticsearch import Elasticsearch** # elasticsearch client sdk

S3 has the ability to trigger an AWS Lambda function whenever a new object is added or deleted, passing to the function’s environment the information, such as the name of the object, bucket in which to object is stored, etc.

**def lambda\_handler(context, event)**: # signature for the lambda function

# lets initialize the AWS Glue client and the elasticsearh client

**glue\_client = boto3.resource**('glue') # we will use glue to get schema information about the data

**es\_client = Elasticsearch([{**

**'host': ELASTICSEARCH\_HOST,**

**'port': ELASTICSEARCH\_PORT**

**}])**

That Lambda function could then go on and add additional data about the object that was created, such as who created it, how long should the object remain in storage, which team the object is available to, the schema of the objects, (since we now have that information available, thanks to AWS Glue) and what is the size of that data.

**glue\_response = glue\_client.get\_table(**

**CatalogId=CATALOG\_ID,** # the id of the data catalog where the data resides

**DatabaseName=DATABASE\_NAME,** # the name of the database in which the table resides

**Name=NAME** # the name of the table for which to retrieve the definition

) # the response of this command is a dict

schema\_data = glue\_response['Table']['StorageDescriptor']

In short, the function could provide metadata about the object that was created.

# now we have to generate a metadata dict to store in elastic search

**metadata\_dict = {**

**'created': datetime.datetime.now(),**

**'owner': 'owner',**

**'bucket': bucket,**

**'filename': filename,**

**'availableTo': ['Advanced Analytics', 'Machine Learning'],**

# additional data can be added as needed

**'schema': schema\_data**

**}**

**##########################################################################**

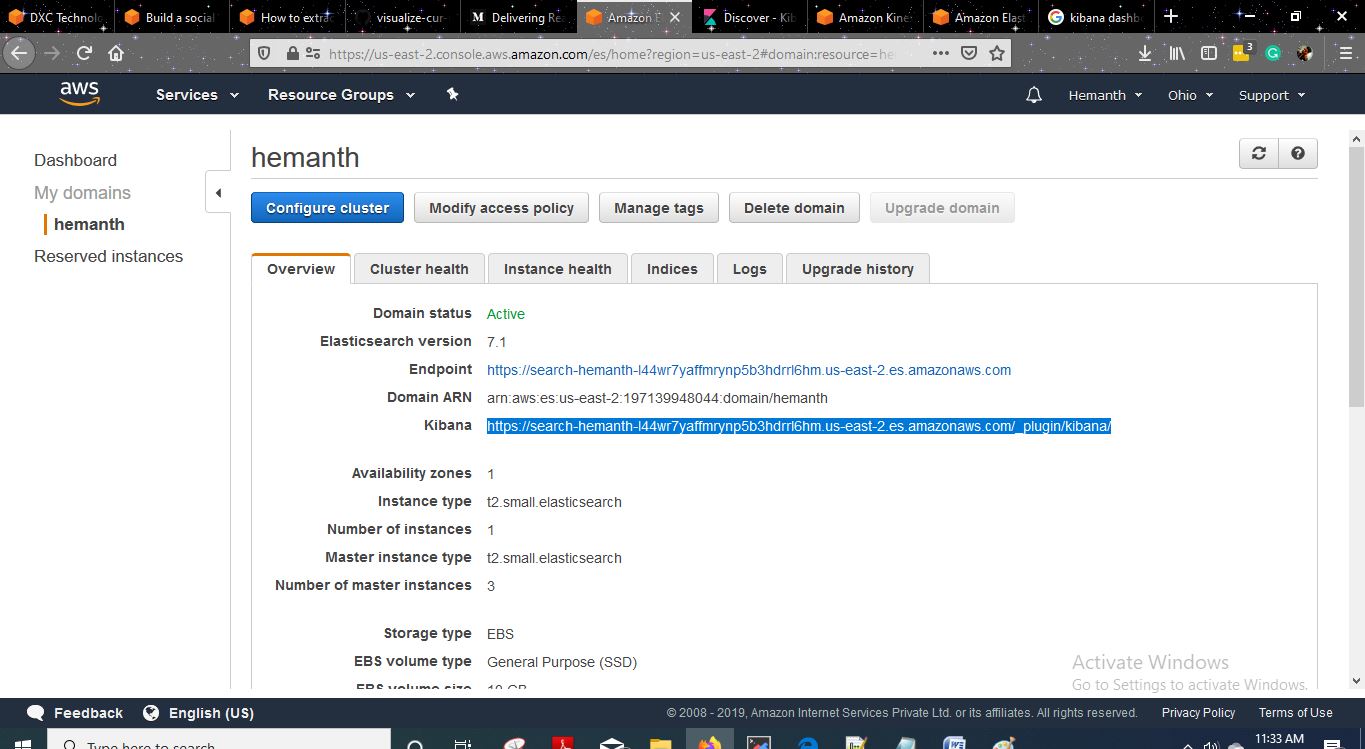
Once you’ve generated the required metadata, you need a place to access that metadata quickly and efficiently. This is the role of the AWS Elasticsearch Service. By enhancing the previously created Lambda function to store the generated metadata inside your Elasticsearch Cluster, you visualize the results of such queries in Kibana. ¸

# what is left now is to save/index data to elasticsearch

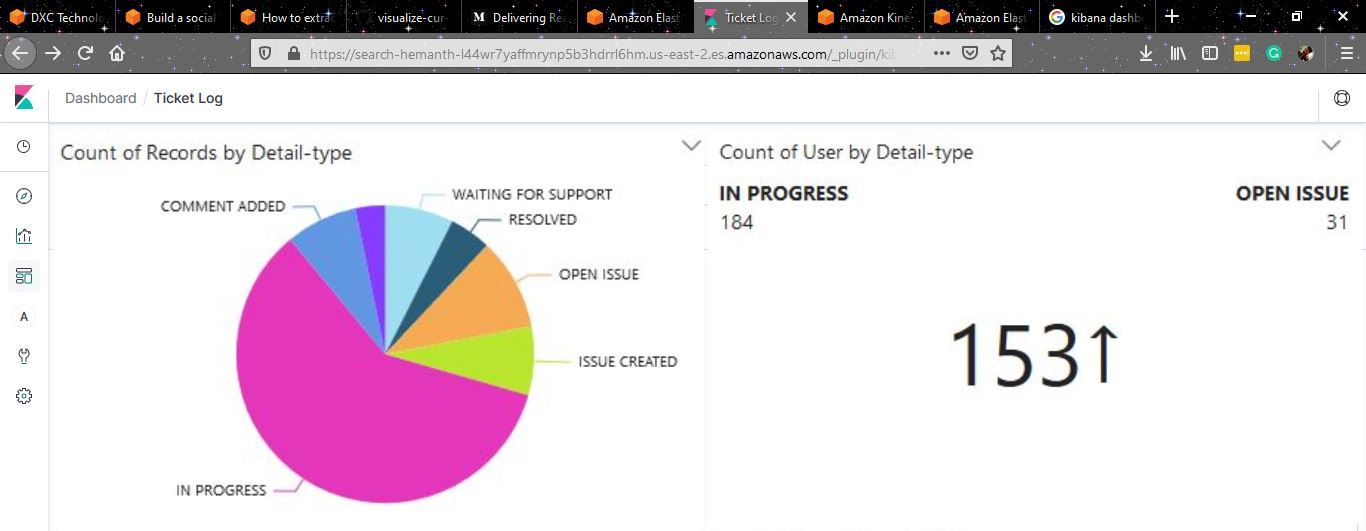
**es\_client.index(index='metadata', doc\_type='metadata', body=metadata\_dict)**

**SEVENTH: Kibana Dashboard**

Open the kibana dashboard from the Elasticsearch console



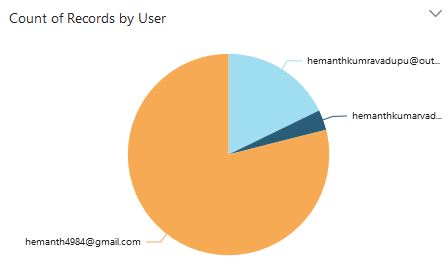
Check the dashboard results of Log analysis of service desk tickets.

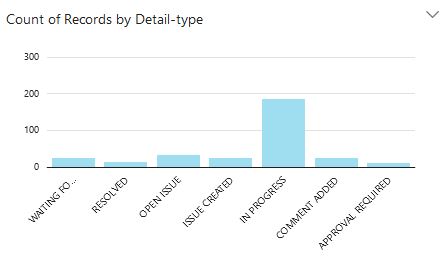


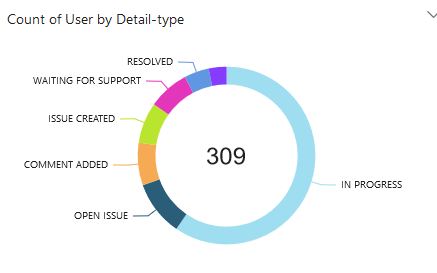
Categorize the logs based on the Issue log events,

1. IN PROGRESS
2. OPEN ISSUE
3. ISSUE CREATED
4. WAITING FOR SUPPORT
5. COMMENT ADDED
6. RESOLVED

You can expand on this dashboard, and build analyses such as this one:







## Conclusion

## This prediction mechanism also can be used as a self-service tool where users can enter ticket descriptions and get back the KM article to solve their own issue. This will also reduce the number of tickets

The entire processing, analytics, and machine learning pipeline starting with Amazon Kinesis, analyzing the data using Amazon Comprehend to perform sentiment analysis and Elastic Search and kibana to create the dashboards was built without spinning up any servers.

I added advanced machine learning (ML) services to our flow, through some simple calls within AWS Lambda, and we built a multi-lingual analytics dashboard with Kibana. We have also saved all the data to Amazon S3 so, if we want, we can do other analytics on the data using Amazon EMR , Amazon SageMaker, Amazon Elasticsearch Service, or other AWS services.