Illinois Institute of Technology

CSP 554 – Big Data Technologies

Customer Churn Prediction using PySpark MlLib and AWS Kinesis

Sai Manohar Vemuri (A20514848)

Sesha Sai Sushmith Muchintala(A20536372)

Syed Wali Uddin Quadri (A20554645)

Prof. Navendu Garg

Table of Contents

Introduction

Recent machine learning and Artificial Intelligence model shifted their paradigm on relying on instant and streaming data from generators like the Internet of Things (IoT). This brings in the necessity of making predictions on continuous data streams instead of static data. Traditional AI/ML frameworks like Python generators have limitations in handling sheer amounts of data and instant processing. These models and frameworks which relied on persistent datasets and static data, became impractical for applications, especially in business operations, stocks, health care, etc. A new approach is laid to tackle this problem using streaming services like Kafka or AWS kinesis, where the continuous data will be streamed from a cloud storage through a streaming service and a machine learning model that is deployed on the endpoint will make predictions for the streamed data and store the results. This also helped in managing and reusing the data streams which resulted in no utilization of data storage/ file systems. This project presents a machine learning modeling on streaming data using AWS tools (a cutting-edge tool that is extensively used in the industry)

Literature Review

The article [1], from AWS documentation demonstrates the process of clustering handwritten digits using the SageMaker PySpark library. The data manipulation is performed through Spark, utilizing a SparkSession. Subsequently, the SageMaker Spark library is employed for interaction with SageMaker, covering both training with K-Means clustering on the MNIST dataset and utilizing pre-existing models for inference. The notebook has been created and validated on an ml.m4.xlarge notebook instance. The SageMakerEstimator.fit() method returns a SageMakerModel, enabling DataFrame transformation by invoking an Amazon SageMaker Endpoint. This process involves serialization of DataFrame rows for prediction requests. The serialization is performed using a RequestRowSerializer, and predictions are deserialized into Spark Rows using a ResponseRowDeserializer. Custom data formats can be supported by implementing your own RequestRowSerializer and ResponseRowDeserializer.

The article [3] explores the integration of Amazon SageMaker, Kinesis Datastream, Kinesis Data Analytics, API Gateway, and Lambda functions for real-time machine learning inference on streaming data. Leveraging a subset of the NYC taxi ride dataset, a SageMaker linear learner is trained and deployed behind an endpoint. The architecture includes a Python data generator script for simulated streaming, while Apache Flink handles data processing. The study showcases a comprehensive framework for dynamic, real-time machine learning inference on streaming data, exemplifying the collective effectiveness of AWS services in this domain.

Kafka ML architecture:

Kafka-ML features a user-friendly front-end powered by Angular, facilitating the management of ML models and configurations. It supports multi-customer distribution and high-rate message dispatching, enabling the concurrent training of multiple PySpark ML models in a streaming environment.

On the back end, a RESTful API is provided for information management, implemented using Django. Kubernetes API is utilized for deployment and administration purposes. For model training, Kubernetes Jobs are employed to train and containerize models, with the results submitted to the back-end.

In the model inference phase, trained models are downloaded from the back-end. These models leverage incoming stream data for predictions, ensuring a balanced data ingestion process through Kafka consumer groups.

A control logger is responsible for sending control messages to the back-end, facilitating easy configuration of data parameters during the deployment of an ML model for inference.

Kafka-ML adopts an efficient deployment and management strategy by deploying Apache Kafka and ZooKeeper as Jobs in Docker containers within the Kubernetes environment, both internally and externally.

Pipeline of PySpark ML model in Kinesis:

Import PySpark MlLib Libraries: Set up the PySpark environment in AWS Sagemaker and import all the necessary libraries.

Model Training: Develop the PySpark model using the sagemaker_pyspark library and train the model using the training data.

Serialization of the Data: The PySpark model needs the serialized data. Common Serialization techniques are RequestRowSerializer, ProtobufRequestRowSerializer, LibSVMRequestRowSerializer, etc.

Deployment: PySpark models built using the sagemaker_pyspark library are automatically deployed after the training and an endpoint is created.

Real-Time Inference: Set up a process to consume real-time data from the Kinesis stream and apply the PySpark ML model for real-time inference. This involves deserializing the incoming data points and feeding them to the model for prediction.

Project Abstract

Our project focuses on the implementation of Churn Risk Prediction using AWS kinesis and machine learning, which allows for real-time ML prediction on streaming data. To accomplish this, we utilized a range of Amazon services, including Amazon SageMaker, Amazon Kinesis stream, Amazon Kinesis Data Analytics, Amazon API Gateway, and a Lambda function.

Our project aims to analyze the streaming data in real-time by building the PySpark ML model and accurately predicting the Customer churn risk, allowing companies to measure the risk of how many customers choose to discontinue using their products. By leveraging some of the machine learning models, we can identify useful patterns and extract useful information. For training this model we will be using the whole dataset comprising of 250,000 datapoints available in Kaggle.

Dataset Description:

The "E-commerce Customer Behaviour and Purchase Dataset" is a carefully built synthetic dataset generated with the Faker Python package, providing a complex and intricate simulation of various customer interactions inside a digital marketplace. This dataset mimics real-world e-commerce dynamics through the inclusion of crucial columns such as Customer ID, Customer Name, Customer Age, Gender, Purchase Date, Product Category, Product Price, Quantity, Total Purchase Amount, Payment Method, Returns, and Churn. It is intended for data analysis and predictive modeling and acts as a diverse resource for e-commerce researchers, data scientists, and analysts. This dataset offers a unique opportunity to study and gain insights into the wide range of e-commerce consumer interactions, thanks to its artificial yet realistic depiction of user behaviors, purchase patterns, and churn indicators.

METHODOLOGY

In this project, real-time predictions are made on streaming data by using AWS tools such as AWS Sagemaker, Kinesis, Cloud9, Lambda function, and Amazon APIs. We have used E-commerce Customer Data and split the data into train, test, and validation datasets which are then stored in an S3 bucket. Following the training of our ML model on AWS Sagemaker using the K-means model, the model is deployed on an endpoint for real-time invoked through APIs and AWS Lamda. The stream of data is generated from an IDE Cloud9, and using Python generator script the data is injected in the Kinesis stream and calls the sagemaker using Flink to make predictions for the streamed data. Results are then stored in the S3 bucket.

Amazon Sagemaker:

Amazon SageMaker[6] is a completely maintained machine learning platform that enables developers and data scientists to rapidly build, train, and launch models using machine learning in a fully functional and operational hosting environment. SageMaker's incorporation of a Jupyter Notebook instance enables simple research and analysis of data sources and eliminates the necessity for server administration. It additionally includes enhanced machine learning algorithms that are capable of handling enormous data sets effectively in a distributed environment. These algorithms cover a wide range of machine-learning tasks, such as regression, classification, clustering, and anomaly detection, among others. SageMaker also supports customized algorithms and frameworks natively, as well as flexible distributed training alternatives adapted to distinctive workflows.

Amazon Kinesis Datastream:

Amazon Kinesis is completely managed by AWS service that helps users collect, analyze, and preserve large data streams in real-time. There are mainly 3 components in the Kinesis framework namely Produces, Streams, and Consumers. Producers help to generate data points continuously while streams are data pipelines that help to process these records. Consumers help to ingest the data via streams and perform some processing on them. Kinesis data streams also integrate with additional

AWS services like AWS Lambda, and S3 which allow building end-to-end real-time data processing pipelines.

Amazon Kinesis Data Analytics is similar to Kinesis where you can process and analyze streaming data using standard SQL. It relies on an open-source stream processing architecture called Apache Flink and helps process data through an array of sources, which includes Kinesis data streams, Kinesis Data Firehouse, and Amazon DynamoDB streams. It has been optimized to be flexible and fault-tolerant. It automatically allocates and expands the number of resources needed for processing the streaming data and can recover from the errors automatically.

Amazon API Gateway:

Amazon API Gateway is used for developing and maintaining Web APIs and can be used for publishing RESTFUL APIs that provide users with access to the backend services, business logic, and data. Various services are integrated with API Gateway such as S3, Lambda, and EC2 which can handle multiple HTTP requests. API Gateway provides various authorization protocols and keys to restrict accessibility. And also offers features like caching, throttling, and request validation to manage and safeguard the APIs.

Amazon Lambda:

Amazon Lambda lets users create and manage apps without worrying about servers. When something happens like someone making an API call or changing the data in storage service (S3), Lambda jumps in and runs the necessary code. It is flexible, understanding various programming languages. For complex apps, it works smoothly with other AWS services and developers don't have to worry about resources as it automatically adjusts the computing resources. It provides an easy way to build, test, and deploy the code, with tools for keeping an eye on how well applications are running.

K-Means Sagemaker Estimator model:

KMeansSagemakerEstimator is a class in Amazon Sagemaker SDK and it is used to train and deploy K-means clustering models. It helps to cluster similar things together and have to specify the number of groups you want. Once trained, it can efficiently organize new data points for these groups. This algorithm works by iteratively assigning data points to the nearest cluster centroids and updating the centroid values. This model can be trained on various data types including

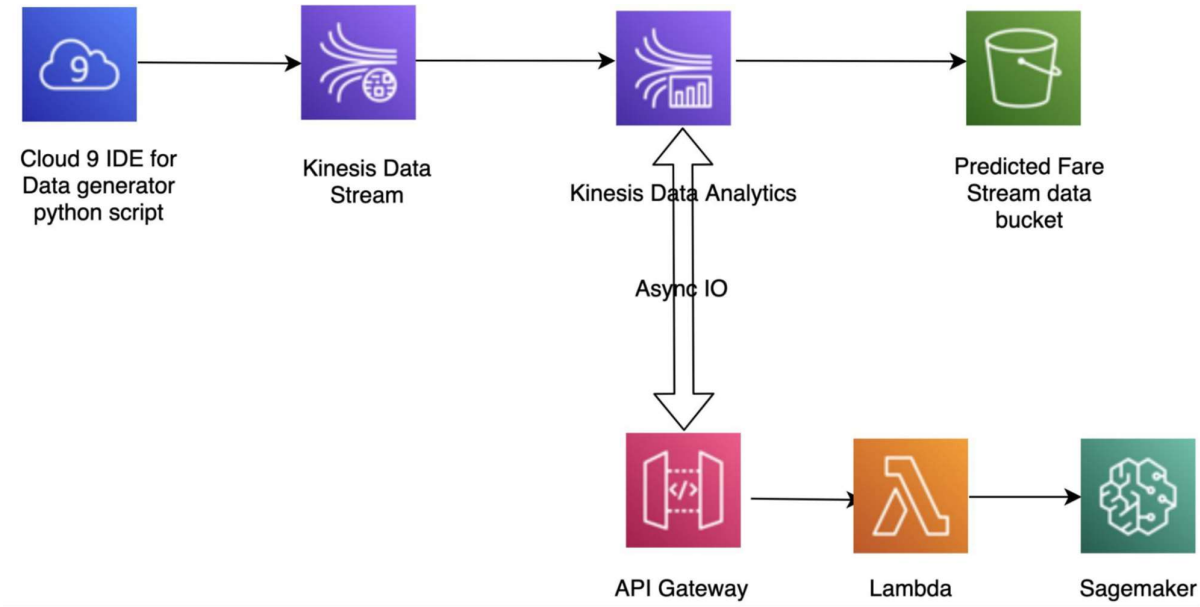
1. CSV files,
2. Parquet files and
3. Amazon record files.

It also simplifies the deploying of the model to Amazon Sagemaker endpoints which can be accessed later for predicting new data.

Implementation

In this project, as outlined in [1], we aim to train a model using a whole dataset with 250,000 data points. However, we have decided to go with a different dataset than the dataset mentioned in [1]. Leveraging the K-means algorithm for machine learning tasks, we'll establish a solid foundation for our model. To make this model operational in real-time scenarios, we'll deploy it behind a SageMaker

endpoint. Real-time e-commerce data will be streamed using a Python data generation tool through Cloud 9. Employing the combination of Apache Flink, Amazon Kinesis Datastream, and Amazon Kinesis Data Analytics services, we'll process the streaming data and trigger the SageMaker endpoint. The setup includes configuring a Java application for Apache Flink in Amazon Kinesis Data Analytics to asynchronously invoke the SageMaker endpoint for incoming streaming data. The processed information, including predicted churn risk, will be stored in an S3 bucket. The overall architecture is meticulously designed for scalability and availability, capable of handling extensive volumes of streaming data.



1) Launching Amazon SageMaker studio

Amazon SageMaker is an IDE for machine learning that helps to build, train, debug, deploy, and monitor machine learning models

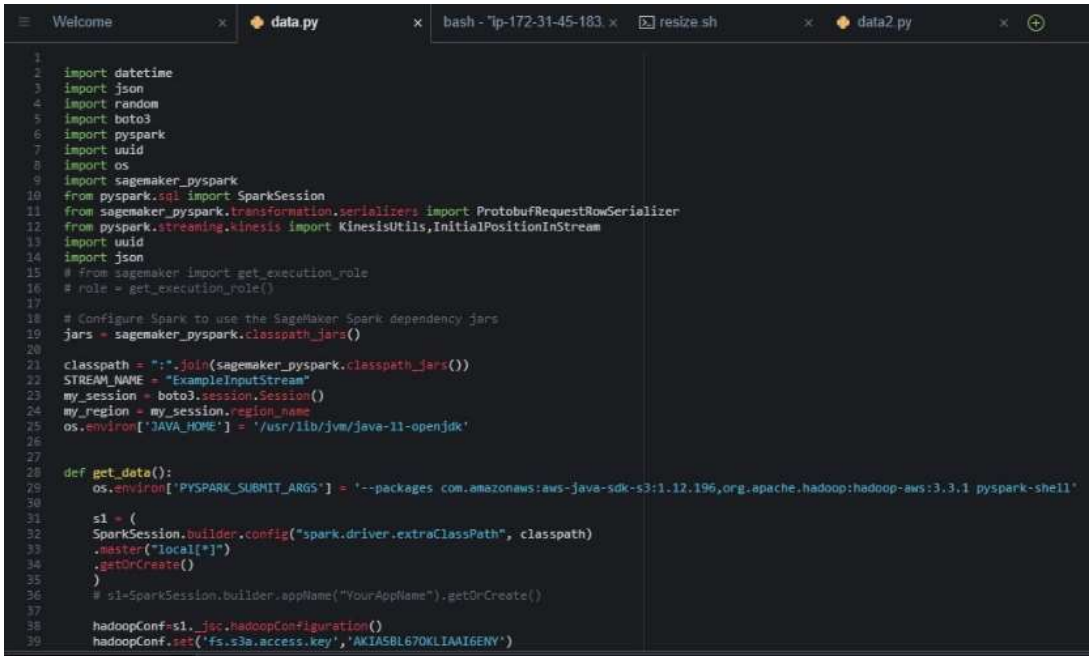
- In the AWS console first select the nearest AWS region. We have chosen N. Virginia.
- Search for Amazon SageMaker in the search Click on Get Started and select Setup SageMaker Domain.
- Enter the domain name and user profile name
- Create a new IAM role under the Execution role. Select the access option “Any S3 bucket”.
- Click on Create Role.

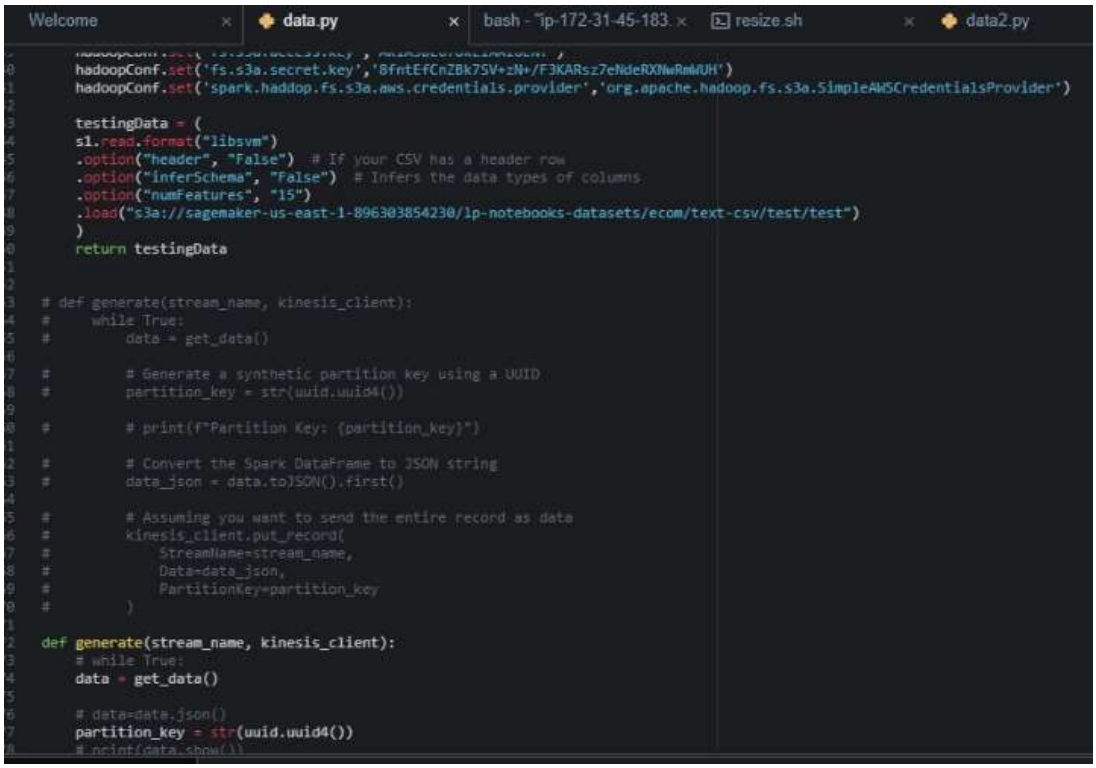
Images

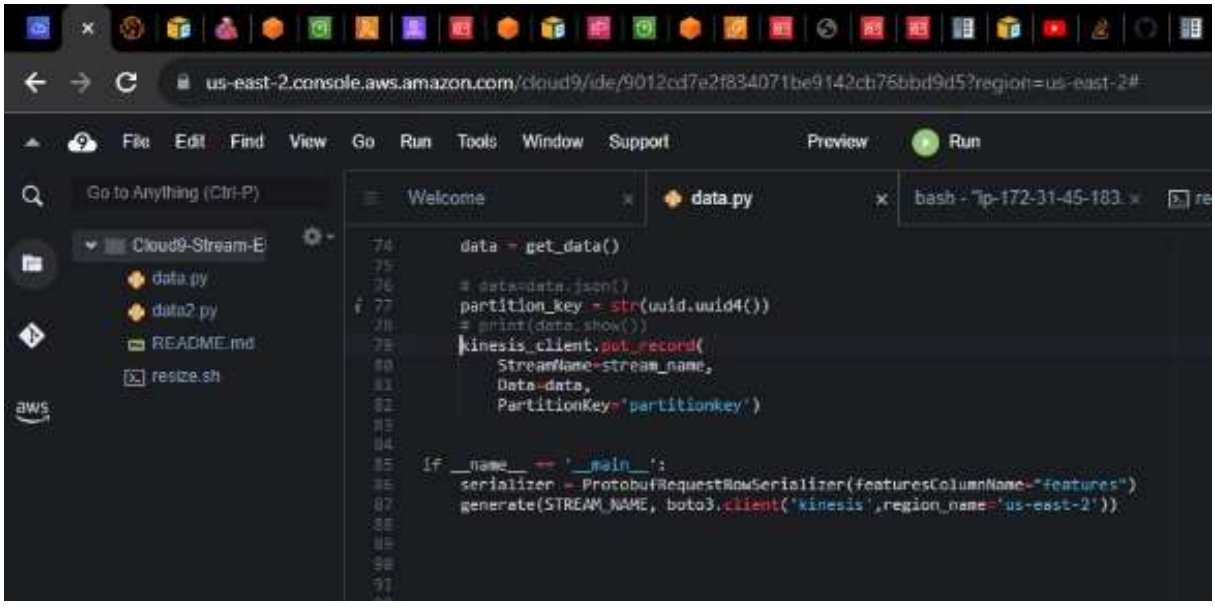
2) Download the Apache Flink Java application jar file:

Amazon SageMaker utilizes the Apache Flink Java application jar file for developing and executing streaming data processing applications. Flink is an open-source distributed platform that allows for efficient processing of real-time data streams. The jar file contains the code and dependencies required to run a Flink application, which can be uploaded to SageMaker to create a job for real-time data processing. With SageMaker's managed Flink environment, users can deploy, operate, and monitor Flink applications without the need to manage the underlying infrastructure.

3) Write data generator streaming Python script:

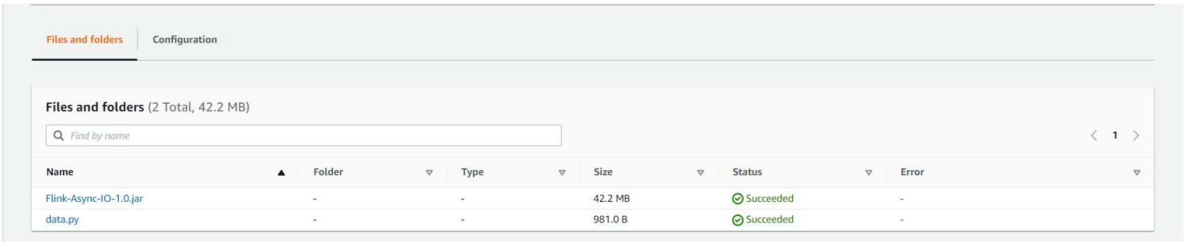



Welcome
data.py
bash - "ip-172-31-45-183.
resize.sh
data2.py
hadoopConf.set('fs.s3a.secret.key','8fntEfCnZBk7SV+zN+/F3KARsz7eNdeRXNwRmWUH')
hadoopConf.set('spark.haddop.fs.s3a.aws.credentials.provider','org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider')
testingData = (
s1.read.format("libsvm")
.option("header", "False") # If your CSV has a header row
.option("inferSchema", "False") # Infers the data types of columns
.option("numFeatures", "15")
.load("s3a://sagemaker-us-east-1-896303854230/1p-notebooks-datasets/ecom/text-csv/test/test")
)
return testingData
def generate(stream_name, kinesis_client):
while True:
data = get_data()
Generate a synthetic partition key using a UUID
partition_key = str(uuid.uuid4())
print(f"Partition Key: {partition_key}")
Convert the Spark DataFrame to JSON string
data_json = data.toJSON().first()
Assuming you want to send the entire record as data
kinesis_client.put_record(
StreamName=stream_name,
Data=data_json,
PartitionKey=partition_key
#)
def generate(stream_name, kinesis_client):
# while True:
data = get_data()
# data=data.json()
partition_key = str(uuid.uuid4())


us-east-2.console.aws.amazon.com/cloud9/ide/9012cd7e2f834071be9142cb76bbd9d5?region=us-east-2#
File Edit Find View Go Run Tools Window Support
Preview
Run
Go to Anything (Ctrl-P)
Cloud9-Stream-E
data.py
data2.py
README.md
resize.sh
Welcome
data.py
bash - "ip-172-31-45-183.
data = get_data()
# data=data.json()
partition_key = str(uuid.uuid4())
# print(data.show())
kinesis_client.put_record(
StreamName=stream_name,
Data=data,
PartitionKey='partitionkey')
if __name__ == '__main__':
serializer = ProtobufRequestRowSerializer(featuresColumnName="features")
generate(STREAM_NAME, boto3.client('kinesis',region_name='us-east-2'))

4) S3 bucket

Upload downloaded Apache Flink Java application jar file and data generator streaming Python script under sage maker studio in s3 bucket.



5) Model training

Upload the dataset to the same directory where we created a jupyter notebook in SageMaker studio.

A) Write a program for training:

```
import os
import boto3
import re
import sagemaker
import numpy as np
```

```
role = sagemaker.get_execution_role()
sess= sagemaker.Session()

region=boto3.Session().region_name
data_bucket=sess.default_bucket()
data_prefix = "lp notebooks-datasets/ecom/text csv"
output_bucket = data_bucket
output_prefix ="sagemaker/DEMO-linear-learner-ecom-regression"
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /root/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /root/.config/sagemaker/config.yaml
```

Install Necessary JAVA libraries for PySpark:

```
# https://github.com/geerlingguy/ansible-role-java/issues/64
!mkdir -p /usr/share/man/man1

# https://stackoverflow.com/a/61902164/4281353
# !apt update -y
!apt-get update -y
!apt install software-properties-common -y
# !apt-add-repository 'deb http://security.debian.org/debian-security stretch/updates main'
!apt install default-jdk -y
# !apt install openjdk-8-jdk -y
!apt-get -y update && apt-get -y upgrade
# !apt-get install -y openjdk-8-jdk
```

[5]

```
Hit:1 http://deb.debian.org/debian bullseye InRelease
Hit:2 http://security.debian.org/debian-security bullseye-security InRelease
Ign:3 http://security.debian.org/debian-security stretch/updates InRelease
Hit:4 http://deb.debian.org/debian bullseye-updates InRelease
```

Set JAVA_HOME:

```
!java --version
!ls '/usr/lib/jvm/'
# %env JAVA_HOME='/usr/lib/jvm/java-1.8.0-openjdk-amd64'

# %env JAVA_HOME='/usr/lib/jvm/java-11-openjdk-amd64'

# import os
os.environ['JAVA_HOME'] = '/usr/lib/jvm/java-11-openjdk-amd64'
```

[6]

```
openjdk 11.0.21 2023-10-17
OpenJDK Runtime Environment (build 11.0.21+9-post-Debian-1deb11u1)
OpenJDK 64-Bit Server VM (build 11.0.21+9-post-Debian-1deb11u1, mixed mode, sharing)
default-java  java-1.11.0-openjdk-amd64  java-11-openjdk-amd64  openjdk-11
```

Upload the dataset to the same directory where we created a jupyter notebook in SageMaker studio and read the file:

```
from sagemaker_pyspark.algorithms import KMeansSageMakerEstimator
from sagemaker_pyspark import SageMakerModel
from pyspark.sql import SparkSession
import sagemaker_pyspark

import os
import pyspark
import sagemaker
from sagemaker import get_execution_role
# os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.amazonaws:aws-java-sdk-pom:1.10.34,org.apache.hadoop:hadoop-aws:2.7.2 pyspark-shell'
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.amazonaws:aws-java-sdk-s3:1.12.196,org.apache.hadoop:hadoop-aws:3.3.1 pyspark-shell'
from pyspark.sql import SQLContext
from pyspark import SparkContext
role = get_execution_role()
jars = sagemaker_pyspark.classpath_jars()

classpath = ":".join(sagemaker_pyspark.classpath_jars())


from pyspark import SparkConf
from pyspark.sql import SparkSession
import sagemaker_pyspark
import botocore.session
```

```
session = botocore.session.get_session()
credentials = session.get_credentials()
from pyspark.sql import SparkSession
spark = (
    SparkSession.builder.config("spark.driver.extraClassPath", classpath)
    .master("local[*]")
    .getOrCreate()
)

hadoopConf=spark._jsc.hadoopConfiguration()
hadoopConf.set('fs.s3a.access.key','AKIA5BL67OKLIAAI6ENY')
hadoopConf.set('fs.s3a.secret.key','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')
hadoopConf.set('spark.haddop.fs.s3a.aws.credentials.provider','org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider')
df = spark.read.csv("ecommerce_customer_data_large.csv", header=True, inferSchema=True, encoding='utf-8')

df.show()
```

Sample of the data frame:

```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
23/11/26 20:06:28 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

+-----------+------------------+----------------+-------------+--------+---------------------+--------------+------------+-------+--------------+---+------+-----+
|Customer ID|      Purchase Date|Product Category|Product Price|Quantity|Total Purchase Amount|Payment Method|Customer Age|Returns| Customer Name|Age|Gender|Churn|
+-----------+------------------+----------------+-------------+--------+---------------------+--------------+------------+-------+--------------+---+------+-----+
|      44605|2023-05-03 21:30:02|            Home|          177|       1|                 2427|        PayPal|          31|    1.0|   John Rivera| 31|Female|    0|
|      44605|2021-05-16 13:57:44|     Electronics|          174|       3|                 2448|        PayPal|          31|    1.0|   John Rivera| 31|Female|    0|
|      44605|2020-07-13 06:16:57|           Books|          413|       1|                 2345|   Credit Card|          31|    1.0|   John Rivera| 31|Female|    0|
|      44605|2023-01-17 13:14:36|     Electronics|          396|       3|                  937|          Cash|          31|    0.0|   John Rivera| 31|Female|    0|
|      44605|2021-05-01 11:29:27|           Books|          259|       4|                 2598|        PayPal|          31|    1.0|   John Rivera| 31|Female|    0|
|      13738|2022-08-25 06:48:33|            Home|          191|       3|                 3722|   Credit Card|          27|    1.0|Lauren Johnson| 27|Female|    0|
|      13738|2023-07-25 05:17:24|     Electronics|          205|       1|                 2773|   Credit Card|          27|   null|Lauren Johnson| 27|Female|    0|
|      13738|2023-02-05 19:31:48|           Books|          370|       5|                 1486|          Cash|          27|    1.0|Lauren Johnson| 27|Female|    0|
|      13738|2021-12-21 03:29:05|            Home|           12|       2|                 2175|          Cash|          27|   null|Lauren Johnson| 27|Female|    0|
|      13738|2023-02-09 00:53:14|     Electronics|           40|       4|                 4327|          Cash|          27|    0.0|Lauren Johnson| 27|Female|    0|
|      33969|2023-02-28 19:58:23|        Clothing|          410|       3|                 5018|   Credit Card|          27|   null|   Carol Allen| 27|  Male|    0|
|      33969|2023-01-05 11:15:27|            Home|          304|       1|                 3883|        PayPal|          27|    1.0|   Carol Allen| 27|  Male|    0|
|      33969|2023-07-18 23:36:50|           Books|           54|       2|                 4187|        PayPal|          27|    0.0|   Carol Allen| 27|  Male|    0|
|      33969|2021-12-20 23:44:57|     Electronics|          428|       4|                 2289|          Cash|          27|    0.0|   Carol Allen| 27|  Male|    0|
|      33969|2020-03-07 21:31:35|           Books|          281|       1|                 3810|          Cash|          27|    0.0|   Carol Allen| 27|  Male|    0|
|      33969|2022-07-21 04:25:44|            Home|          193|       2|                 3198|   Credit Card|          27|    0.0|   Carol Allen| 27|  Male|    0|
|      33969|2023-07-05 15:01:04|        Clothing|          473|       3|                 2881|   Credit Card|          27|    1.0|   Carol Allen| 27|  Male|    0|
|      42650|2020-10-18 23:38:52|           Books|          127|       5|                 3347|          Cash|          20|    0.0| Curtis Smith| 20|Female|    0|
|      42650|2020-05-17 17:02:36|            Home|          284|       2|                 3531|   Credit Card|          20|    1.0| Curtis Smith| 20|Female|    0|
|      42650|2022-03-18 13:52:08|     Electronics|          256|       2|                 3548|   Credit Card|          20|    0.0| Curtis Smith| 20|Female|    0|
+-----------+------------------+----------------+-------------+--------+---------------------+--------------+------------+-------+--------------+---+------+-----+
only showing top 20 rows
```

6)Data Preprocessing:

We noticed that our dataset contains feature columns like Customer ID, name, Age, and Purchase Date which are irrelevant to our task. So we have decided to drop them.

```
[4]
df1=df.drop(*['Customer ID','Customer Name','Customer Age','Purchase Date'])
df1.show()
```

And as part of data cleaning, we made sure that there were no NaN values in the dataset before passing it to the model for training. As we can see in the Returns column there are few NaN values and we filled it with the default value “0”.

```
[15]
summary_df=df1.describe()

summary_df.filter(summary_df["summary"]=="count").show()
```

```
...
+-------+----------------+-------------+--------+---------------------+--------------+-------+------+------+------+
|summary|Product Category|Product Price|Quantity|Total Purchase Amount|Payment Method|Returns|   Age|Gender| Churn|
+-------+----------------+-------------+--------+---------------------+--------------+-------+------+------+------+
|  count|          250000|       250000|  250000|               250000|        250000| 202618|250000|250000|250000|
+-------+----------------+-------------+--------+---------------------+--------------+-------+------+------+------+
```

```
[17]  df1 = df1.fillna(0)
```

We observed that our dataset contains columns with categorical features. so we have decided to perform One-hot-encoding for these specific columns: **Product Category, Gender, and Payment Method**. The reason we did this is that these columns do not contain ordinal data i.e. no order preference. The following is the code to perform one-hot-encoding and after converting our dataframe contains 14 features and 1 target variable.

```
from pyspark.sql.functions import udf,col
from pyspark.sql.types import IntegerType
import numpy as np

prod_categories = df1.select('Product Category').distinct().rdd.flatMap(lambda x : x).collect()
gender_categories = df1.select('Gender').distinct().rdd.flatMap(lambda x : x).collect()
payment_categories = df1.select('Payment Method').distinct().rdd.flatMap(lambda x : x).collect()
prod_categories.sort()
gender_categories.sort()
payment_categories.sort()
for category in prod_categories:
    function = udf(lambda item: 1 if item == category else 0, IntegerType())
    new_column_name = 'Product Category'+'_'+category
    df1 = df1.withColumn(new_column_name, function(col('Product Category')))

for category in gender_categories:
    \
    function = udf(lambda item: 1 if item == category else 0, IntegerType())
    new_column_name = 'Gender'+'_'+category
    df1 = df1.withColumn(new_column_name, function(col('Gender')))

for category in payment_categories:
    function = udf(lambda item: 1 if item == category else 0, IntegerType())
    new_column_name = 'Payment Method'+'_'+category
    df1 = df1.withColumn(new_column_name, function(col('Payment Method')))
df1=df1.drop(*['Product Category','Payment Method','Gender'])
df1_cols=df1.drop("Churn").columns
df1_cols=np.array(df1_cols)
df1_cols = np.append(df1_cols, "Churn")
df1_cols=list(df1_cols)
df1 = df1.selectExpr(*[f"`{col}`" for col in df1_cols])
df1.show(truncate=False)
```

```
[18]
[Stage 19:>                                                 (0 + 1) / 1]
+-------------+--------+-----------------------+-------+---+-----------------------+-------------------------+-----------------------------+-----------------------+-------------+-------------+-----------------------+-------------------------------+-------------------------+-----+
|Product Price|Quantity|Total Purchase Amount  |Returns|Age|Product Category_Books |Product Category_Clothing|Product Category_Electronics |Product Category_Home  |Gender_Female|Gender_Male  |Payment Method_Cash    |Payment Method_Credit Card     |Payment Method_PayPal    |Churn|
+-------------+--------+-----------------------+-------+---+-----------------------+-------------------------+-----------------------------+-----------------------+-------------+-------------+-----------------------+-------------------------------+-------------------------+-----+
|177          |1       |2427                   |1.0    |31 |0                      |0                        |0                            |1                      |1            |0            |0                      |0                              |1                        |0    |
|174          |3       |2448                   |1.0    |31 |0                      |0                        |1                            |0                      |1            |0            |0                      |0                              |1                        |0    |
|413          |1       |2345                   |1.0    |31 |1                      |0                        |0                            |0                      |1            |0            |0                      |1                              |0                        |0    |
|396          |3       |937                    |0.0    |31 |0                      |0                        |1                            |0                      |1            |0            |1                      |0                              |0                        |0    |
|259          |4       |2598                   |1.0    |31 |1                      |0                        |0                            |0                      |1            |0            |0                      |0                              |1                        |0    |
|191          |3       |3722                   |1.0    |27 |0                      |0                        |0                            |1                      |1            |0            |0                      |1                              |0                        |0    |
|285          |1       |2773                   |0.0    |27 |0                      |0                        |1                            |0                      |1            |0            |0                      |1                              |0                        |0    |
|370          |5       |1486                   |1.0    |27 |1                      |0                        |0                            |0                      |1            |0            |1                      |0                              |0                        |0    |
|12           |2       |2175                   |0.0    |27 |0                      |0                        |0                            |1                      |1            |0            |1                      |0                              |0                        |0    |
|40           |4       |4327                   |0.0    |27 |0                      |0                        |1                            |0                      |1            |0            |1                      |0                              |0                        |0    |
|410          |3       |5018                   |0.0    |27 |0                      |1                        |0                            |0                      |0            |1            |0                      |1                              |0                        |0    |
|304          |1       |3883                   |1.0    |27 |0                      |0                        |0                            |1                      |0            |1            |0                      |0                              |1                        |0    |
|54           |2       |4187                   |0.0    |27 |1                      |0                        |0                            |0                      |0            |1            |0                      |0                              |1                        |0    |
|428          |4       |2289                   |0.0    |27 |0                      |0                        |1                            |0                      |0            |1            |1                      |0                              |0                        |0    |
|281          |1       |3810                   |0.0    |27 |1                      |0                        |0                            |0                      |0            |1            |1                      |0                              |0                        |0    |
|193          |2       |3198                   |0.0    |27 |0                      |0                        |0                            |1                      |0            |1            |0                      |1                              |0                        |0    |
|473          |3       |2881                   |1.0    |27 |0                      |1                        |0                            |0                      |0            |1            |0                      |1                              |0                        |0    |
|127          |5       |3347                   |0.0    |20 |1                      |0                        |0                            |0                      |1            |0            |1                      |0                              |0                        |0    |
|284          |2       |3531                   |1.0    |20 |0                      |0                        |0                            |1                      |1            |0            |0                      |1                              |0                        |0    |
|256          |2       |3548                   |0.0    |20 |0                      |0                        |1                            |0                      |1            |0            |0                      |1                              |0                        |0    |
+-------------+--------+-----------------------+-------+---+-----------------------+-------------------------+-----------------------------+-----------------------+-------------+-------------+-----------------------+-------------------------------+-------------------------+-----+
only showing top 20 rows
```

Converting to Feature vectors:

PySpark's VectorAssembler is used to combine selected columns from the train_data and test_data DataFrames into a single feature vector column named "features," preparing the data for machine learning model training and testing.

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
columns_to_convert = [
    "Product Price", "Quantity", "Total Purchase Amount", "Returns", "Age",
    "Product Category_Books", "Product Category_Clothing", "Product Category_Electronics",
    "Product Category_Home", "Gender_Female", "Gender_Male",
    "Payment Method_Cash", "Payment Method_Credit Card", "Payment Method_PayPal", "Churn"
]

label_column = "Churn"

assembler = VectorAssembler(
    inputCols=columns_to_convert,
    outputCol="features"
)

train_data = assembler.transform(train_data)
test_data = assembler.transform(test_data)

train_data = train_data.select(label_column, "features")
test_data = test_data.select(label_column, "features")
```

[410]

Write to S3 bucket:

We are saving the train_data DataFrame in the LIBSVM format, overwriting any existing data, and storing it in the "train" directory in the S3 bucket. LIBSVM format is commonly used for efficient representation of sparse datasets in machine learning due to its simplicity, space efficiency, and compatibility with various libraries and platforms.

We can use this data for training the model and also for prediction from Kinesis data streams.

```
train_data.write.format("libsvm").mode("overwrite").save("train")
```

[349]

...

```
test_data.write.format("libsvm").mode("overwrite").save("test")
```

[547]

...

Read Train and Test Data from the S3 bucket in LIBSVM format:

```
# trainingData = (
#     spark.read.format("libsvm")
#     .option("numFeatures", "14")
#     .load("s3a://sagemaker-us-east-1-548775606164/1p-notebooks-datasets/taxi/text-csv/train/train.csv".format(region))
# )
# s3://sagemaker-studio-896303854230-2oke5zb2q4i/sagemaker/DEMO-linear-learner-ecom-regression

trainingData = (
    spark.read.format("libsvm")
    .option("header", "True")  # If your CSV has a header row
    .option("inferSchema", "False")  # Infers the data types of columns
    .option("numFeatures", "15")
    .load("s3a://sagemaker-us-east-1-896303854230/1p-notebooks-datasets/ecom/text-csv/train/train")
)
```

[385]

```
testingData = (
    spark.read.format("libsvm")
    .option("header", "True")  # If your CSV has a header row
    .option("inferSchema", "False")  # Infers the data types of columns
    .option("numFeatures", "15")
    .load("s3a://sagemaker-us-east-1-896303854230/1p-notebooks-datasets/ecom/text-csv/test/test")
)
```

[521]

7)Training Data Schema:

```
trainingData.printSchema()
```

[386]

```
root
 |-- label: double (nullable = true)
 |-- features: vector (nullable = true)
```

```
testingData.printSchema()
```

[523]

```
root
 |-- label: double (nullable = true)
 |-- features: vector (nullable = true)
```

Load Necessary Libraries for Building the Model:

We are going to use the KMeansSageMakerEstimator Model.

```
from sagemaker_pyspark import SageMakerEstimator
from sagemaker_pyspark.transformation.serializers import ProtobufRequestRowSerializer,RequestRowSerializer,UnlabeledCSVRequestRowSerializer
from sagemaker_pyspark.transformation.deserializers import ProtobufResponseRowDeserializer
from sagemaker_pyspark import IAMRole
from sagemaker_pyspark import RandomNamePolicyFactory,RandomNamePolicy
from sagemaker_pyspark import EndpointCreationPolicy
```

[394]

```
from sagemaker.amazon.amazon_estimator import get_image_uri
container = sagemaker.image_uris.retrieve("linear-learner", sess.boto_region_name)
```

[360]

We are configuring a SageMaker KMeans estimator, setting parameters such as the SageMaker role, training and endpoint instance types, feature dimension, and the number of clusters (K) for the KMeans algorithm.

Here since we have 2 classes, we choose K=2. And no.of features = 15 including the target variable.

```
[400]
# from sagemaker_pyspark.transformation.serializers import CSVSerializer
# from sagemaker_pyspark.transformation.deserializers import UnlabeledCSVRequestRowSerializer
from sagemaker_pyspark.transformation.serializers import UnlabeledCSVRequestRowSerializer
endpoint_config_name = "my-kmeans-endpoint-config"
kmeans_estimator = KMeansSageMakerEstimator(
    sagemakerRole=IAMRole(role),
    trainingInstanceType="ml.m4.xlarge",  # Instance type to train K-means on SageMaker
    trainingInstanceCount=1,
    endpointInstanceType="ml.t2.large",  # Instance type to serve model (endpoint) for inference
    endpointInitialInstanceCount=1
)
# kmeans_estimator._set(endpointConfigName=endpoint_config_name)
kmeans_estimator.setFeatureDim(15)
kmeans_estimator.setK(2)
# kmeans_estimator.setTrainingInput(recordIOData=csv_format_data, contentType="text/csv", serializer=CSVSerializer)
```

Train the model:

After creating the Estimator classes, the instances we requested are provisioned and configured with the necessary libraries. Then, the data is downloaded from our channels into the instance. Subsequently, the training process commences. After the training process is complete, the trained model is automatically deployed on Amazon SageMaker's real-time hosted endpoint. This will help to generate predictions from the model. We did not host the model on the same type of instance used for training because training is a computationally expensive task that has different computation and memory requirements from hosting. Hence, we can choose a different instance type for hosting the model. We trained our model on an ml.m4.xlarge instance, but we used ml.t2.large to host the model because this instance is a less expensive CPU instance.

```
[401]
customModel = kmeans_estimator.fit(trainingData)

...
23/11/26 23:06:12 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
23/11/26 23:06:12 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
23/11/26 23:06:12 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
```

8)Prediction:

We tested our model on test data using the model. transform function and the following is the output. The closest cluster column represents the predicted label 0 or 1. And label column represents the Actual Label.

```
[427]
    transformedData = customModel.transform(testingData)

    transformedData.show()
```

```
[Stage 161:>                                                        (0 + 1) / 1]
+-----+--------------------+------------------+---------------+
|label|            features|distance_to_cluster|closest_cluster|
+-----+--------------------+------------------+---------------+
|  1.0|(15,[0,1,2,3,4,7,...|  1085.364013671875|            0.0|
|  0.0|(15,[0,1,2,3,4,7,...|   594.557373046875|            0.0|
|  1.0|(15,[0,1,2,3,4,8,...|  1025.041259765625|            0.0|
|  0.0|(15,[0,1,2,4,7,9,...| 1197.7076416015625|            0.0|
|  0.0|(15,[0,1,2,4,5,9,...| 1002.5492553710938|            1.0|
|  1.0|(15,[0,1,2,3,4,8,...| 251.60684204101562|            1.0|
|  0.0|(15,[0,1,2,4,5,10...|  705.8165283203125|            1.0|
|  1.0|(15,[0,1,2,4,7,9,...| 1368.0238037109375|            1.0|
|  0.0|(15,[0,1,2,3,4,8,...|   1039.66357421875|            0.0|
|  0.0|(15,[0,1,2,4,8,10...|     695.9775390625|            0.0|
|  0.0|(15,[0,1,2,3,4,7,...|  822.4688720703125|            1.0|
|  0.0|(15,[0,1,2,4,7,10...|  530.9755249023438|            1.0|
|  0.0|(15,[0,1,2,3,4,6,...|  494.2064208984375|            1.0|
|  0.0|(15,[0,1,2,4,7,10...|  473.4258117675781|            1.0|
|  0.0|(15,[0,1,2,4,8,10...|  675.2708740234375|            0.0|
|  1.0|(15,[0,1,2,3,4,6,...|    533.76025390625|            0.0|
|  0.0|(15,[0,1,2,3,4,6,...| 397.04974365234375|            0.0|
|  0.0|(15,[0,1,2,3,4,8,...| 1084.3748779296875|            0.0|
|  0.0|(15,[0,1,2,3,4,5,...|      1232.77734375|            1.0|
|  0.0|(15,[0,1,2,4,5,10...| 1089.0753173828125|            1.0|
+-----+--------------------+------------------+---------------+
only showing top 20 rows
```

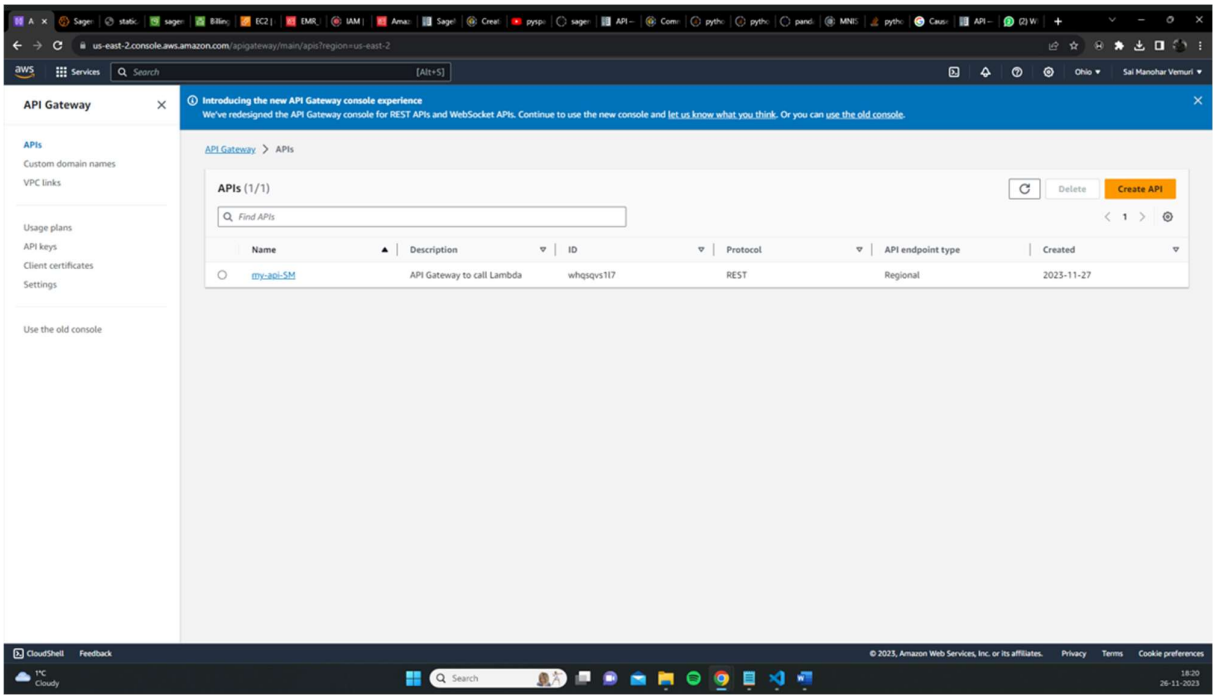
Check the Endpoint name:

We will use this endpoint for the Kinesis Datastream.

```
[403]
    initialModelEndpointName=customModel.endpointName
```

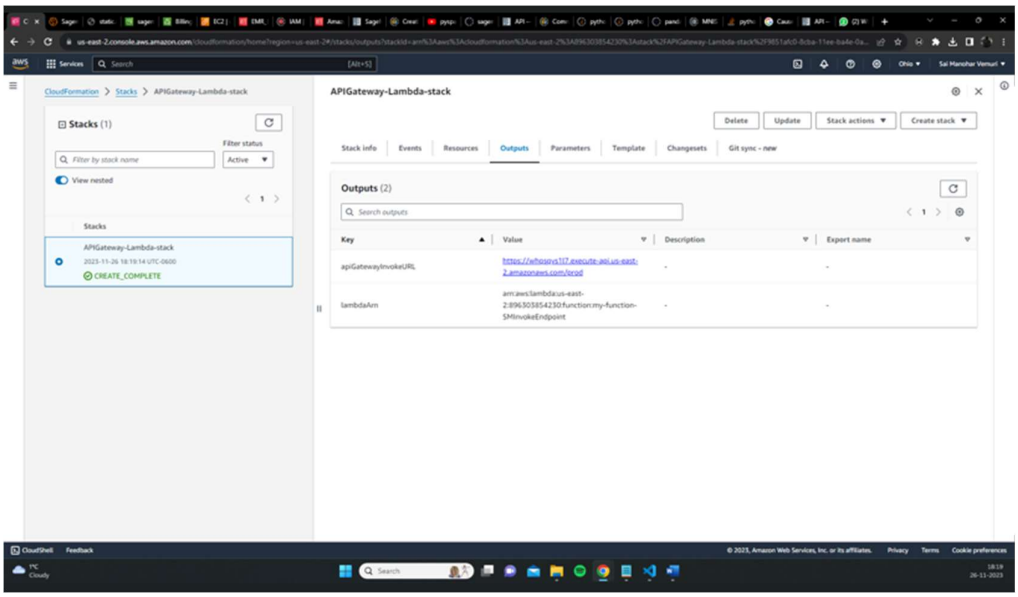
```
[448]
    ENDPOINT_NAME = initialModelEndpointName
    print(ENDPOINT_NAME)

endpoint-0d524ba8b028-2023-11-26T23-06-12-155467
```

9)Creating APIGateway-Lambda-Stack:-



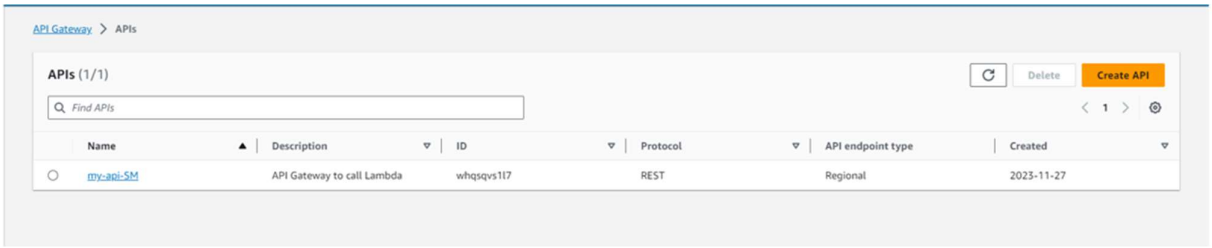
- This is done to create an API Gateway and Lambda Function.
- In the sage maker endpoint the created python .ipynb file is entered.
- The API gateway attributes are declared
- A Lambda function has been created to call a SageMaker endpoint and make predictions for churn This Lambda function is then called by API Gateway to obtain the predicted churn for streaming data that is being sent from Kinesis

APIGateway-Lambda-stack



The created API gateway and Lambda function have been initialized.

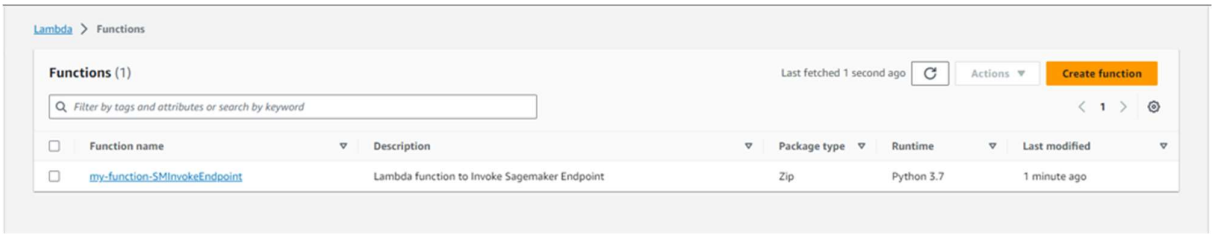
Verify the created API gateway function and lambda function



allow the Lambda function's API Gateway endpoint to be accessed by the AWS Kinesis Data Analytics Apache Flink application.

● Getting into the AWS API Gateway Service console and going to the Lambda function-corresponding API Gateway instance.

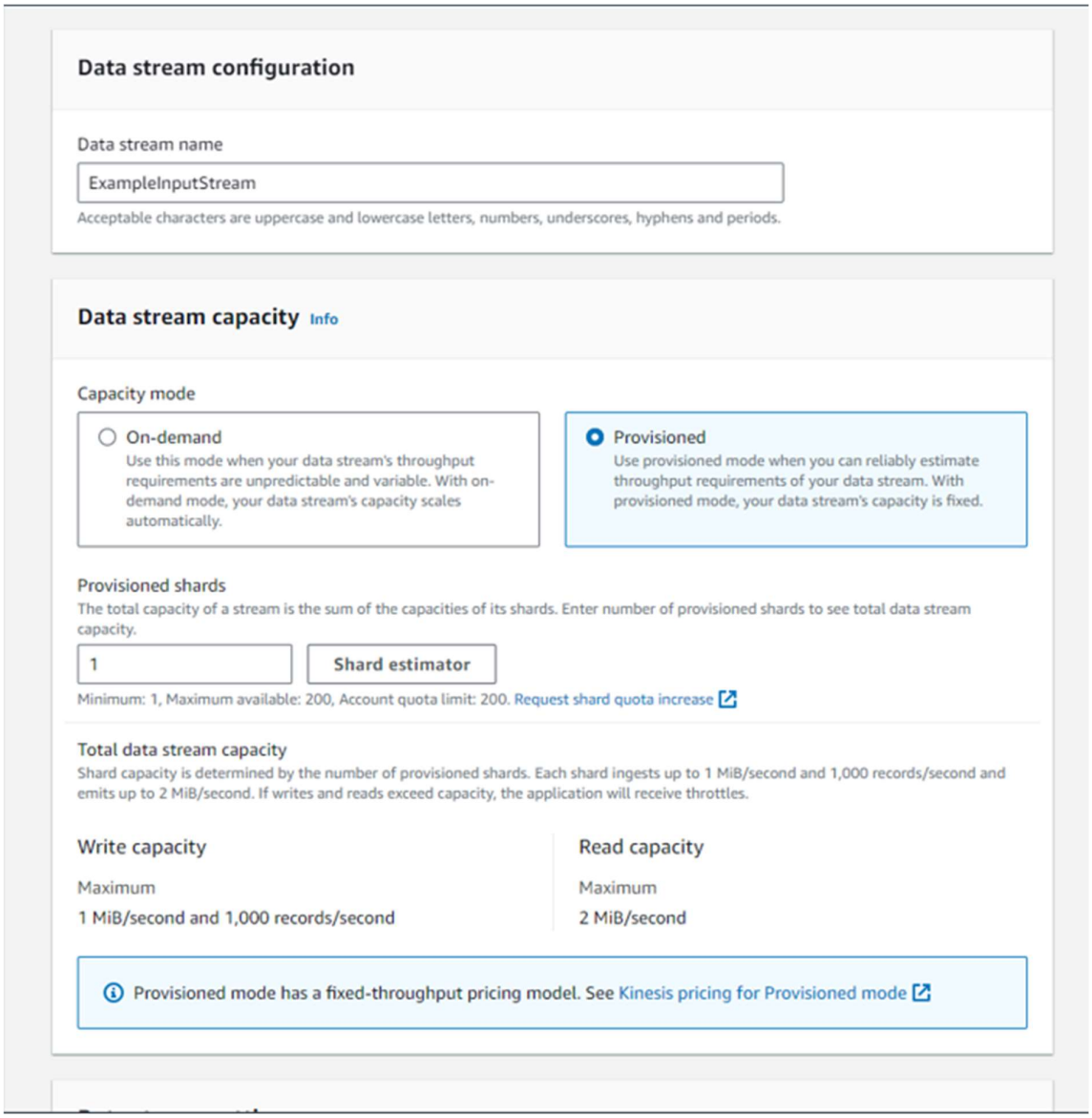
● The 'prod' stage invoke URL is stored for future reference. Using this URL, the Kinesis Data Analytics application will call the APGateway endpoint as an ASync I/Oendpoint.



examining the Lambda and API functions that have been created in the corresponding application window.

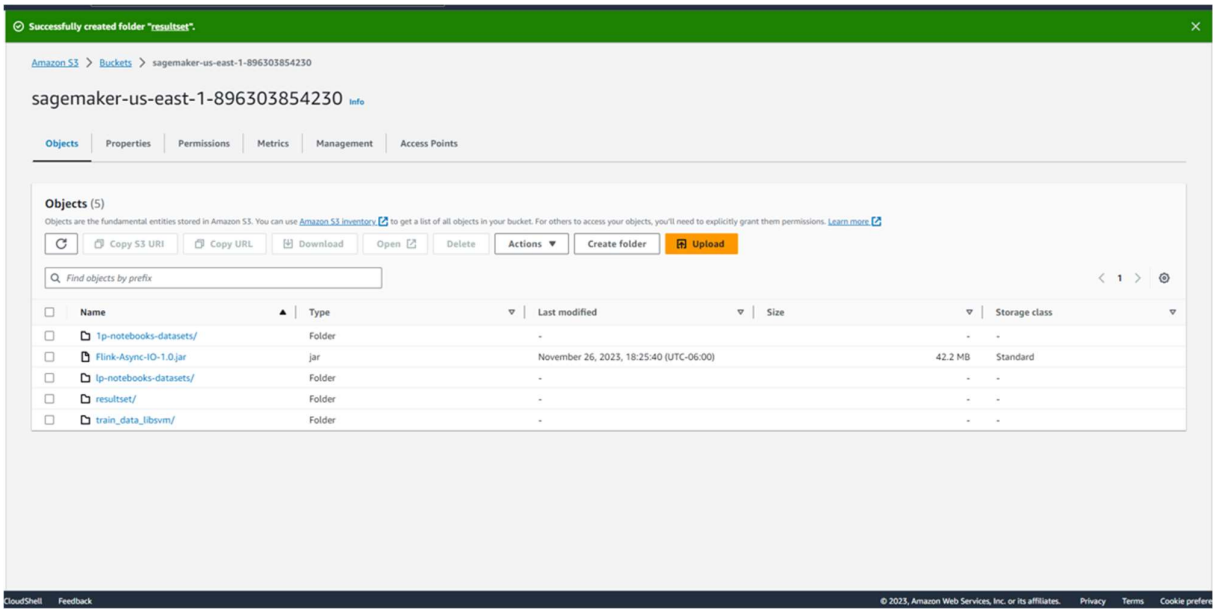
10)Creation of Data Streaming Kinesis:-

The Amazon Kinesis Data Stream Service in the AWS console is where a Kinesis data stream is created. The data stream capacity is set to provision, and its name is set to "ExampleInput Stream." One provisioned shard is the only number.

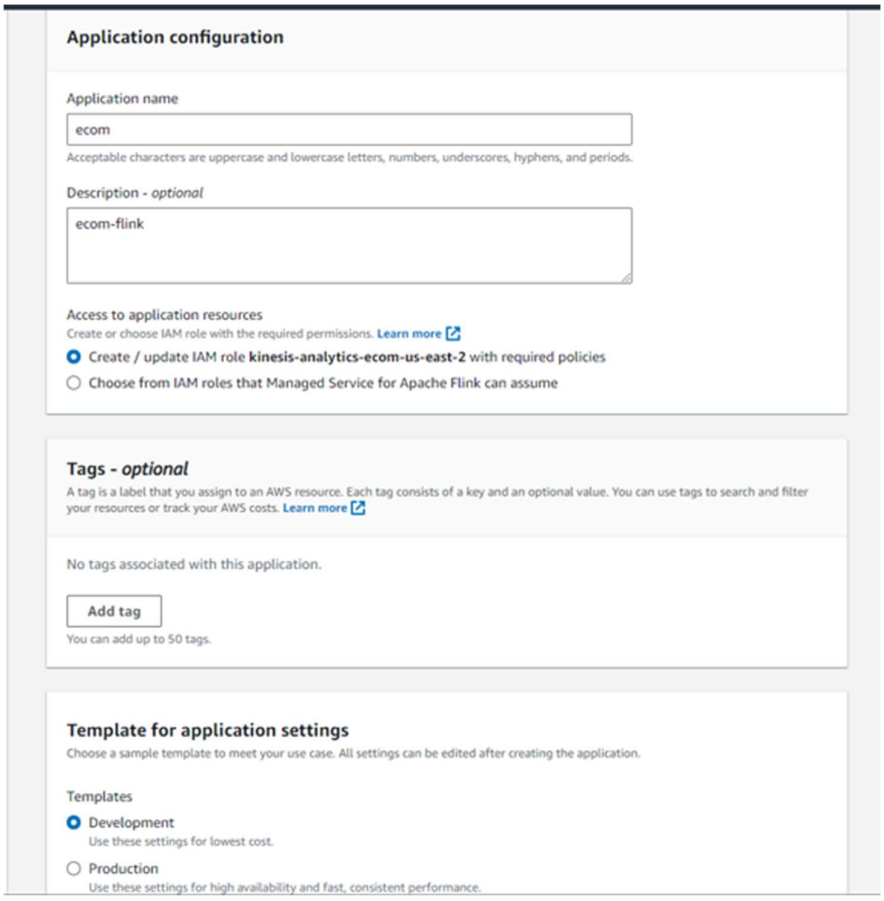


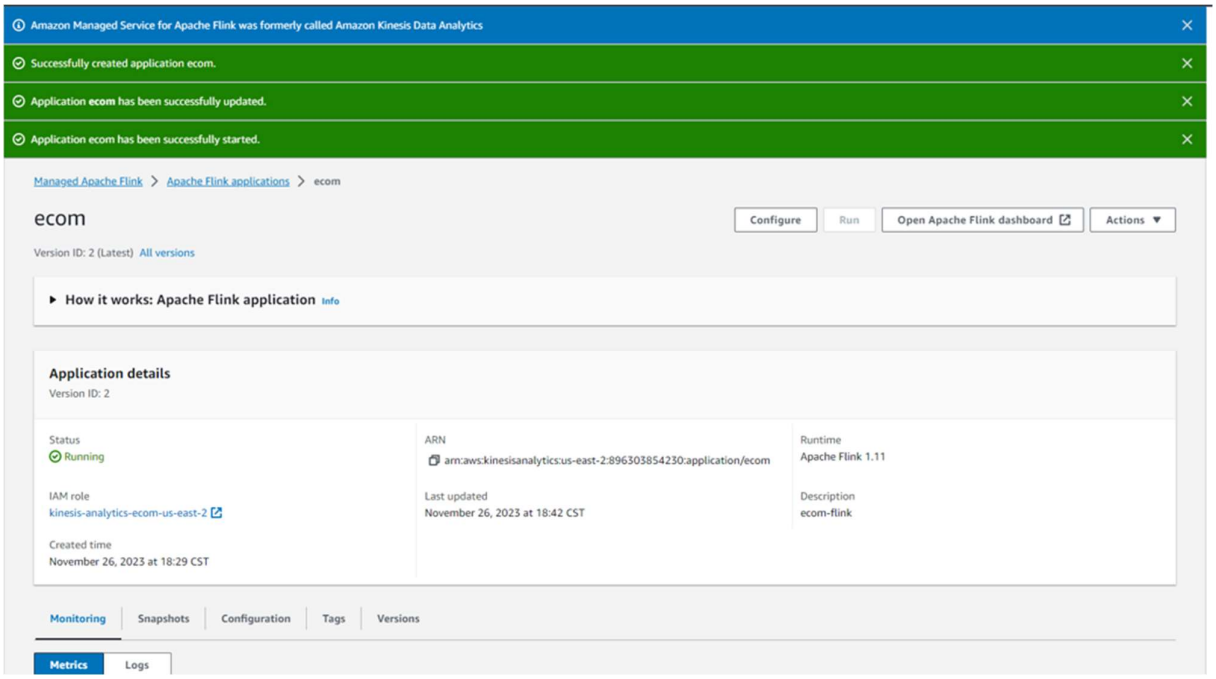
Construct Kinesis data analytics (an application for Apache Flink):-

In the Sagemaker Studio bucket on the S3 console, a folder named "resultset" is created. Since the Kinesis Data Analytics Application (KDA) would store all of the inference results in the S3 bucket, the S3 bucket was created before the KDA. The PageMaker endpoint generates the inference results, which are then asynchronously input/output returned to the KDA. The KDA will receive the path of the "result" folder to store the results. This Apache Flink application's jar file is available in an S3 bucket.



The Java application for Apache Flink is made. To help with the real-time incoming data stream churn prediction, Apache Flink will leverage its Async I/Operator feature. To obtain the churn prediction, the API Gateway is activated asynchronously for every data record. Next, the predictions and the associated data record are saved in the S3 bucket. The recommended version of Apache Flink, 1.11, is the one that is selected. The application is identified as "com." Due to its low cost, the template was selected for "development."



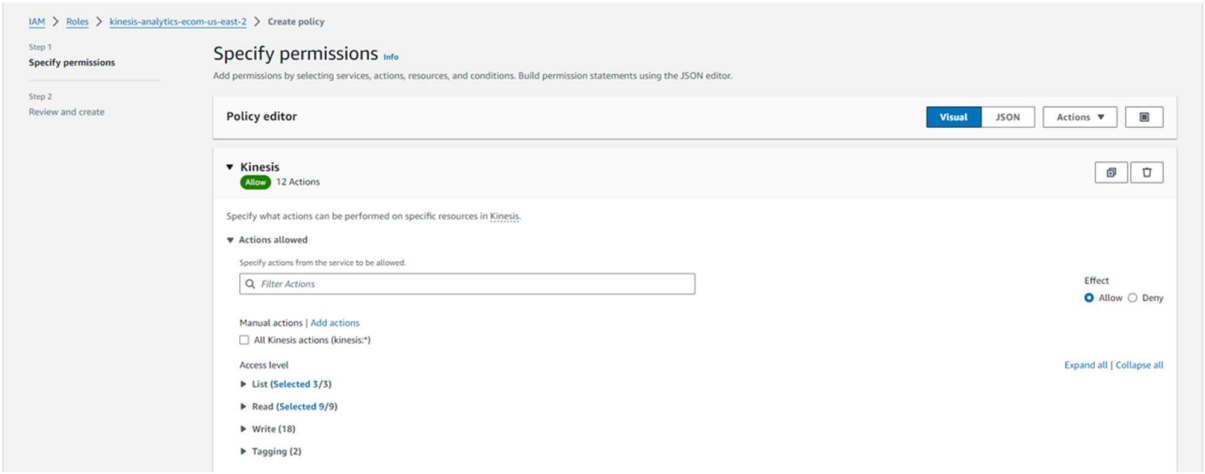


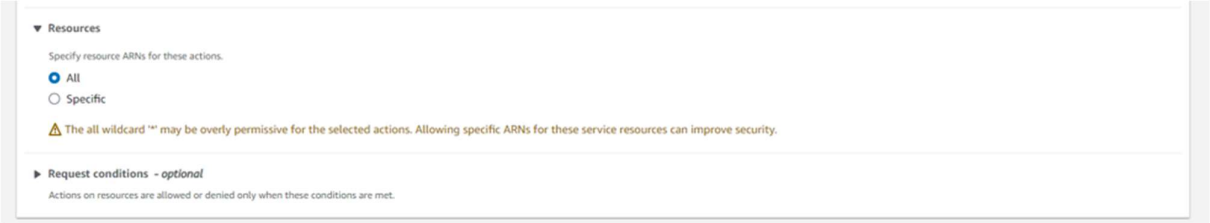
The "ecom" application needs to be configured next. The name of the S3 bucket is entered under Amazon S3 Bucket, and the name of the jar file is entered under Path to S3 Object.

A total of our key-value pairs are created, and the Group ID has been created as Flink Application Properties in the runtime properties section. In each column, the names of the keys are listed along with the values that correspond to them.

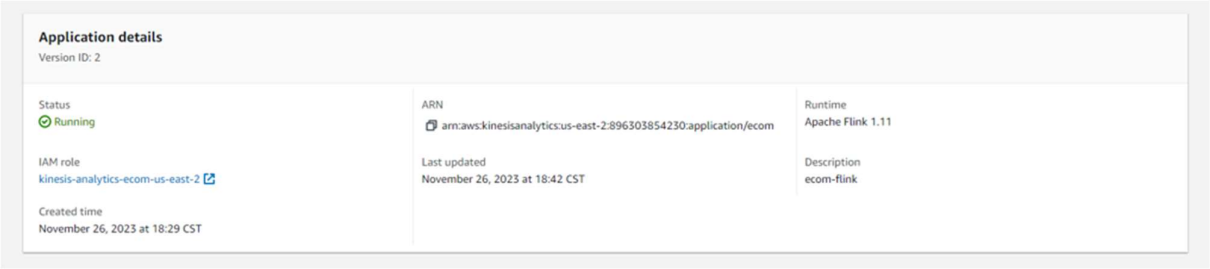
Give appropriate IAM policies and proceed.

The KDA application does not currently have the ability to list and read permissions. Additionally, it lacks the authorization to list or read from the S3 bucket. Thus, this application needs to have a few more IAM policies added. List and Read are chosen under "access levels" after the kinesis service has been chosen in order to grant the permissions under "create inline policies." All actions and resources have now been chosen for S3.

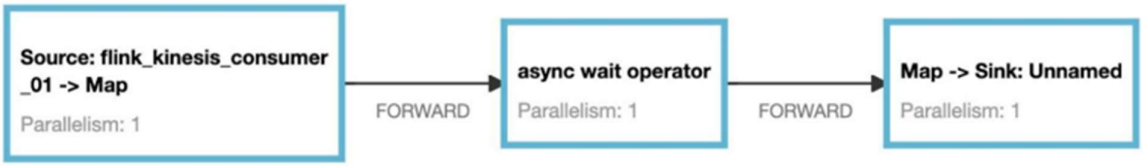




The policy needs to be created and given a name in the following step. The "RunWithout Snapshot" option is then selected to launch the Kinesis application. The application has been successfully launched after being run.

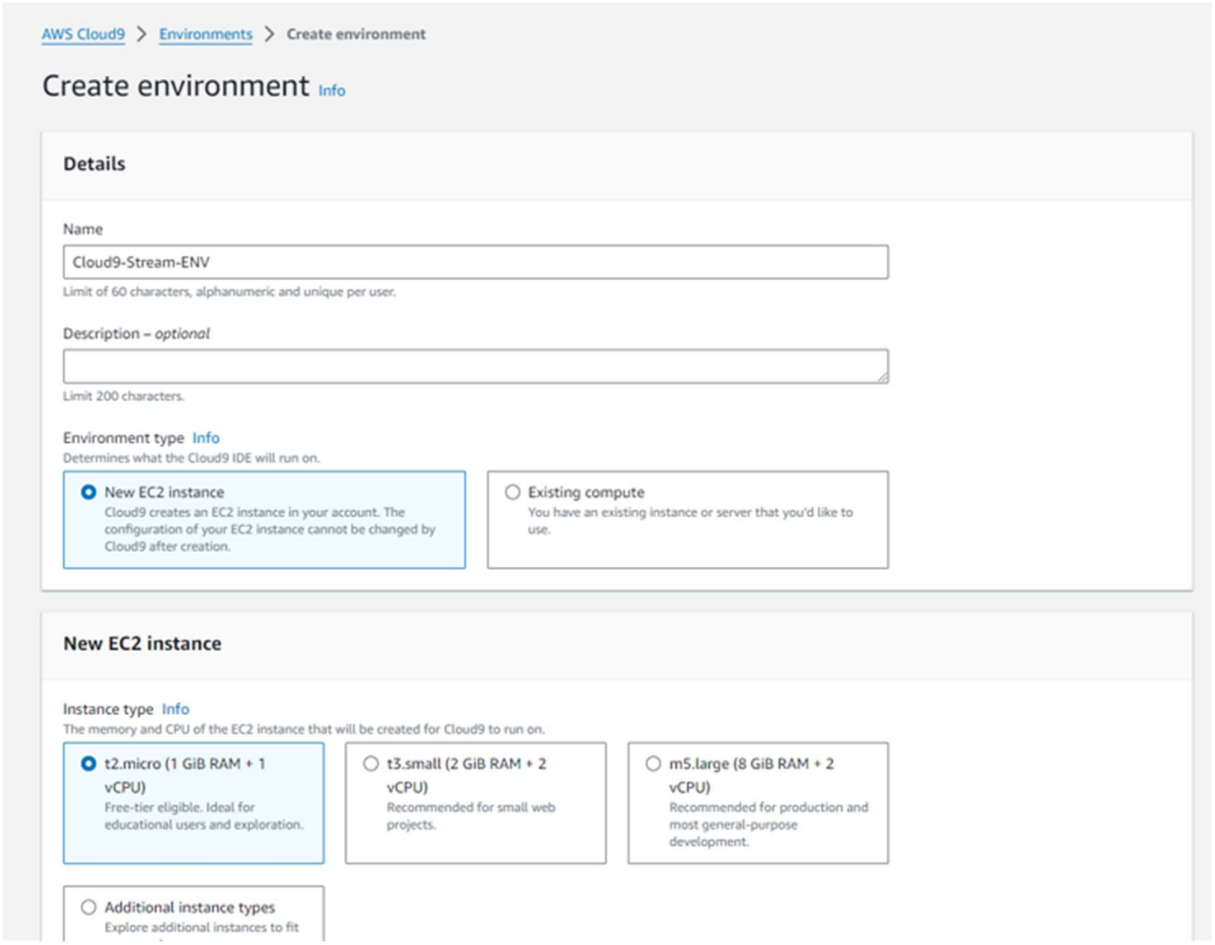


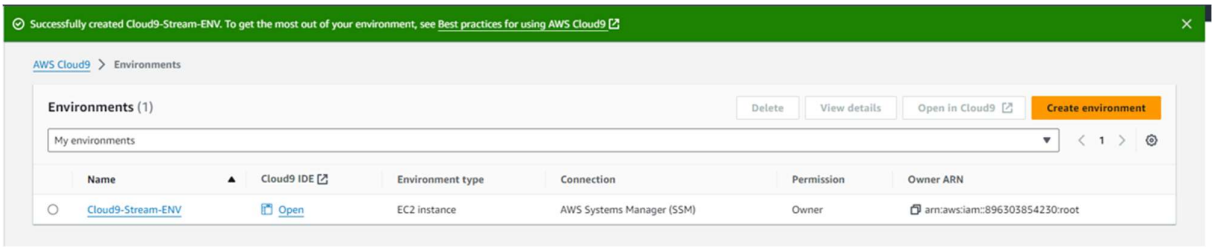
Below is the application graph. A visual depiction of the dataflow made up of the operators and intermediate results is called an application graph.



The Python data generation code (data.py) for creating streaming data is stored in the S3 bucket. It selects at random and sends the streaming data to the previously established Kinesis data stream. Once the streaming data enters the data stream, it will be ingested by the Kinesis data analytics application. The resulting dataset, which contains the churn predicted values, is then stored in the S3 bucket "resultset" after the API Gateway has been called.

Entire default value set is selected to create the Cloud 9 environment.





We can be taken to the Cloud9 IDE (Integrated Development Environment) after the environment has been created.

The IDE includes multiple panels for file management, command execution, and output viewing in addition to a code editor and terminal.

The data.py file is copied to a newly created file, and it is then saved.

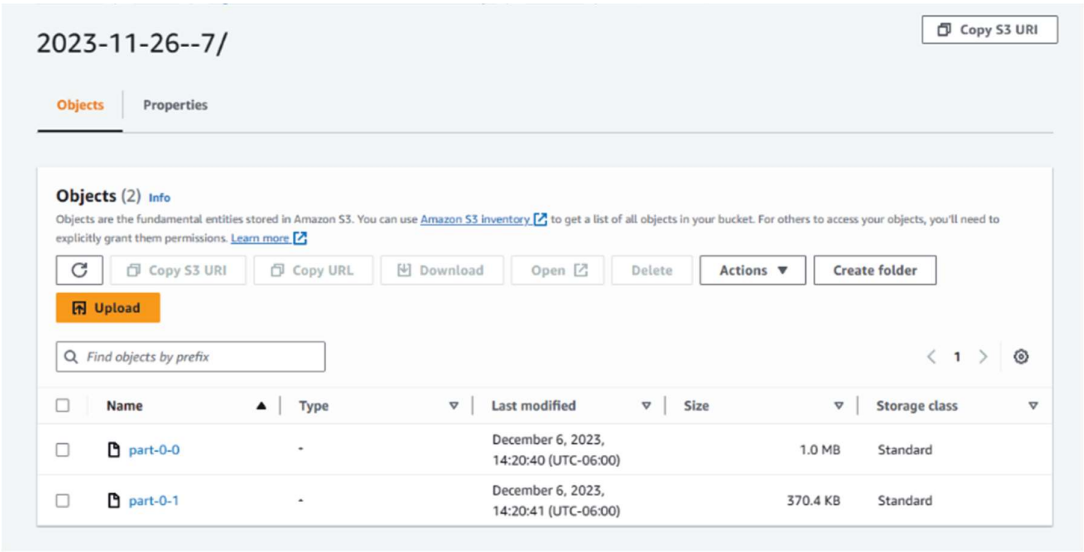
```python
import datetime
import json
import random
import boto3
import pyspark
import uuid
import os
import sagemaker_pyspark
from pyspark.sql import SparkSession
from sagemaker_pyspark.transformation.serializers import ProtobufRequestRowSerializer
from pyspark.streaming.kinesis import KinesisUtils,InitialPositionInStream
import uuid
import json
# from sagemaker import get_execution_role
# role = get_execution_role()

# Configure Spark to use the SageMaker Spark dependency jars
jars = sagemaker_pyspark.classpath_jars()

classpath = ":".join(sagemaker_pyspark.classpath_jars())
STREAM_NAME = "ExampleInputStream"
my_session = boto3.session.Session()
my_region = my_session.region_name
os.environ['JAVA_HOME'] = '/usr/lib/jvm/java-11-openjdk'


def get_data():
    os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.amazonaws:aws-java-sdk-s3:1.12.196,org.apache.hadoop:hadoop-aws:3.3.1 pyspark-shell'

    s1 = (
    SparkSession.builder.config("spark.driver.extraClassPath", classpath)
    .master("local[*]")
    .getOrCreate()
    )
    # s1=SparkSession.builder.appName("YourAppName").getOrCreate()

    hadoopConf=s1._jsc.hadoopConfiguration()
    hadoopConf.set('fs.s3a.access.key','AKIA5BL67OKLIAAI6ENY')
    hadoopConf.set('fs.s3a.secret.key','8fntEfCnZBk7SV+zN+/F3KARsz7eNdeRXNwRmWUH')
    hadoopConf.set('spark.haddop.fs.s3a.aws.credentials.provider','org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider')

    testingData = (
    s1.read.format("libsvm")
    .option("header", "False")  # If your CSV has a header row
    .option("inferSchema", "False")  # Infers the data types of columns
    .option("numFeatures", "15")
    .load("s3a://sagemaker-us-east-1-896303854230/lp-notebooks-datasets/ecom/text-csv/test/test")
    )
    return testingData
```

```python
    data = get_data()

    # data=data.json()
    partition_key = str(uuid.uuid4())
    # print(data.show())
    kinesis_client.put_record(
        StreamName=stream_name,
        Data=data,
        PartitionKey='partitionkey')


if __name__ == '__main__':
    serializer = ProtobufRequestRowSerializer(featuresColumnName="features")
    generate(STREAM_NAME, boto3.client('kinesis',region_name='us-east-2'))
```

Following the code's execution in the terminal, random streaming data is produced. After some time, the creation of streaming data is halted.

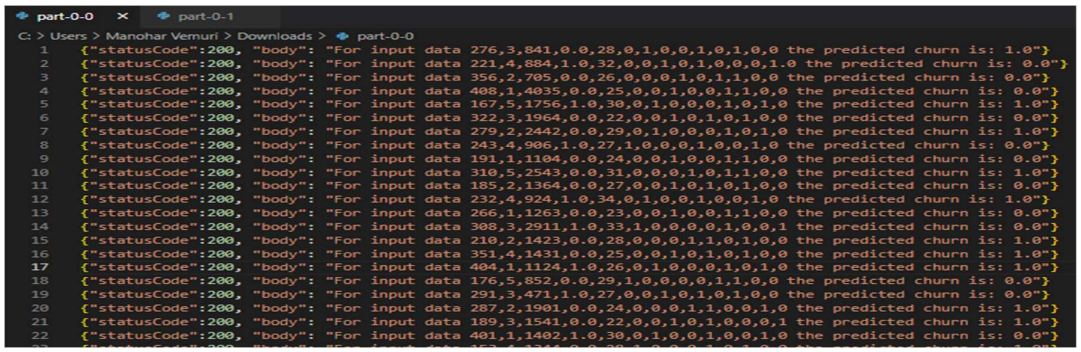
Selecting the configuration to add the code to the Apache Flink application is the next step. Below are the specifics of each of the three positions.

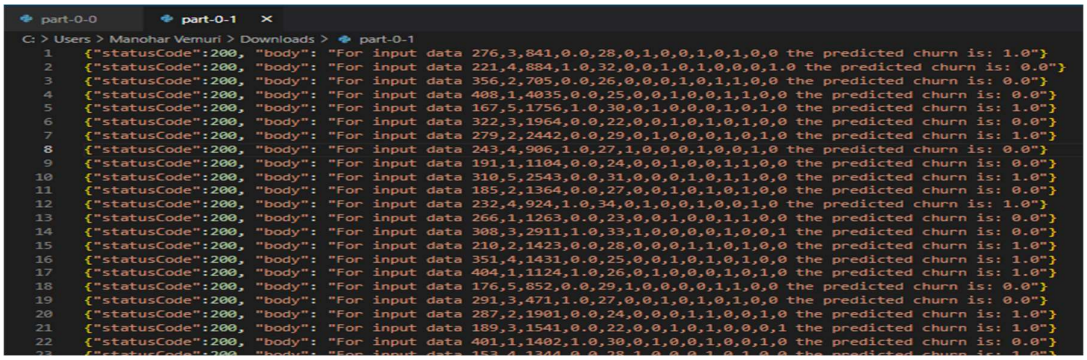
Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parall	Tasks
Source: flink_kinesis_consumer_01 -> Map	RUNNING	0 B	0	0 B	0	1	1
async wait operator	RUNNING	300 B	0	0 B	0	1	1
Sink: Unnamed	RUNNING	300 B	0	0 B	0	1	1



11)Output

The model that is deployed at the endpoint is used to compute the predicted churn for streaming data, and the results are stored in an S3 bucket. This file can be opened using any text editor.





Conclusion

Our project dives into the realm of real-time machine learning predictions on streaming data, leveraging AWS services such as Amazon SageMaker, Amazon Kinesis, Cloud9, Lambda function, and API Gateway. The primary goal was to develop a robust churn risk prediction model for streaming data, allowing businesses to proactively analyze and predict customer churn. We utilized a synthetic dataset sourced from Kaggle, comprising various customer behavioral attributes crucial for churn prediction and customer analysis.

Throughout our project, we successfully navigated the complexities of handling streaming data and deploying machine learning models on live streams. Leveraging the K-Means algorithm via Amazon SageMaker, we trained a model on a substantial dataset, enabling us to predict churn risks effectively. Additionally, we constructed a comprehensive architecture encompassing Apache Flink, Amazon Kinesis Datastream, and Amazon Kinesis Data Analytics to handle the processing and analysis of the streaming data. Furthermore, the project encountered challenges in data preprocessing, feature selection, and model deployment.

In essence, our project represents a practical approach to churn prediction in real-time streaming scenarios, showcasing the efficiency of AWS services in deploying and managing machine learning models for continuous data streams. The solution not only addresses customer churn prediction but also lays the groundwork for broader applications in real-time analytics and predictive modeling in diverse industries.

References:

[1]. https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-spark/pyspark_mnist/pyspark_mnist_kmeans.html

[2]. https://www.kaggle.com/datasets/shriyashjagtap/e-commerce-customer-for-behavior-analysis/

[3]. https://catalog.us-east-1.prod.workshops.aws/workshops/63069e26-921c-4ce1-9cc7-dd882ff62575/en-US/lab7