Brainstorm

* Análisis de sentimientos
* Opiniones de productos, amazon, yelp
* API de Amazon Comprehend (para análisis de texto en general no sólo análisis de productos de Amazon, pero se paga por uso y en este caso haremos el análisis de las opiniones en Google Cloud)
* API de Amazon Product Reviews (Específico para opiniones de productos de Amazon, gratis para vendedores de Amazon, creando una cuenta de desarrollador de Amazon, pero también hay posibilidad de utilizar la API de Outscraper y registrarse como Outscraper y no como desarrollador <https://outscraper.com/>)
* Amazon prohíbe el uso de la API para recopilar datos de opiniones de productos de forma automatizada si no se respetan sus límites: limita la cantidad de datos que puede recopilar de la API por día,limita la cantidad de consultas que puede realizar a la API por segundo
* Elasticsearch es una buena herramienta para análisis de datos, más económica que Cloud Dataproc pero para conjuntos de datos pequeños o medianos
* Cloud Dataproc más costosa que elasticsearch pero mejor para datos en tiempo real o volúmenes grandes
* Para que se ejecuten los jobs uno detrás de otro se puede hacer mediante temas añadiendo código en los scripts que creen el Pub/Sub y que se suscriban si se quiere ejecutar después de generado un tema o se pueden crear triggers para que se ejecute una función detrás de otra
* API de Fakespot: utiliza un algoritmo basado en aprendizaje automático para identificar las reseñas falsas

Diseño del DAaaS

### **Definición la estrategia del DAaaS**

Guardar en una Base de Datos SQL el resultado del análisis de opiniones de los productos de un vendedor de Amazon, para que puedan ser consumidos por los departamentos de ventas, atención al cliente y marketing.

Para ello, recopilaremos las opiniones de nuestros productos de la Api de Amazon y también utilizaremos datos extraídos de la Api de Fakespot para descartar las opiniones falsas.

Una vez limpio el texto, pasaremos un algoritmo de análisis de sentimiento para que nos clasifique las opiniones como positivas, negativas o neutras para tres categorías: precio, calidad, atención al cliente

Y el resultado del análisis lo almacenaremos en una tabla para que pueda ser consumido por los distintos departamentos.

Los objetivos de este análisis son:

* **Mejorar los productos y servicios** ya que dan información sobre los puntos fuertes y débiles del producto.
* **Mejorar la experiencia del cliente** ya que la información puede utilizarse para localizar puntos de mejora en la resolución de los problemas y por tanto mejorar la satisfacción del cliente.
* **Desarrollar nuevas estrategias de marketing**, se pueden identificar necesidades o tendencias del mercado, o realizar los cambios de estrategia oportunos para un mejor posicionamiento frente al competidor.

**Arquitectura DAaaS**

Fuentes de datos:

* Scrapper de la Api de Amazon Product Reviews
* Scrapper de la API de Fakespot

Componentes:

* **Google Cloud Storage**: Para almacenar los archivos.
* **Cloud Dataproc**: Para procesar y analizar los datos.
* **Dataproc Cluster:** Para ejecutar los trabajos.
* **Google Cloud SQL**: Para almacenar el resultado del análisis
* **Cloud Functions**:

-Scrapping de la API de Amazon y Fakespot.

-Cruzar dos archivos y filtrar las opiniones que no sean falsas.

-Limpiar el texto (eliminar signos de puntuación, caracteres no alfabéticos, palabras vacías, normalizar el texto, dividir el texto en tokens, lematizar el texto).

-Ejecutar un algoritmo de clasificación (por ejemplo, Naive Bayes) con Spark MLlib.

* **Pub/Sub Code**: Para organizar el flujo de trabajo y asegurar que las funciones en la nube se ejecuten de manera secuencial
* **Scheduler**: Para programar los trabajos

### **DAaaS Operating Model Design and Rollout**

Tenemos que tener en cuenta que Amazon limita los accesos y en número de reseñas descargadas, así como el número de descargas por segundo, por lo tanto en el Scheduler se programará una vez al día, en el script para el Scrapper ya van limitadas las descargas y se establece un intervalo de tiempo para respetar el número de descargas por segundo.

1. Crear un **proyecto en Google Cloud** y habilitar Cloud Dataproc
2. Crear un **Bucket** en Google Cloud Storage para almacenar los ficheros
3. Crear un **Cluster** de Dataproc
4. Crear un **Cloud Function** con el script de el **Scrapper** de la Api de Amazon y el código **Pub/Sub** para organizar el flujo de trabajo y que las cloud functions se ejecuten de manera secuencial y programar mediante Scheduler *([ejemplo Scrapper](#ttvo9vskvptz) en apartado Desarrollo de la plataforma DAaaS).*

*Nos traemos los campos:*

"Title",

"Text",

"Rating",

"Date",

"ASIN", (id\_producto)

“Review\_id”, (id de la reseña)

"ReviewerId",

"ReviewerName",

"ReviewerLocation"

1. Cargar el CSV resultado del Scrapper en **Cloud Storage**
2. Crear un **Cloud Function** con el script del **Scrapper** de la Api de Fakespot y añadir el código **Pub/Sub** para organizar el flujo de trabajo y que las cloud functions se ejecuten de manera secuencial y programar mediante Scheduler *([ejemplo Scrapper](#v9buubyb2759) en apartado Desarrollo de la plataforma DAaaS).*

*Nos traemos los campos:*

“product\_id “

“review\_id “

“rating” (Este lo creamos con el resultado, si es “F” es falsa)

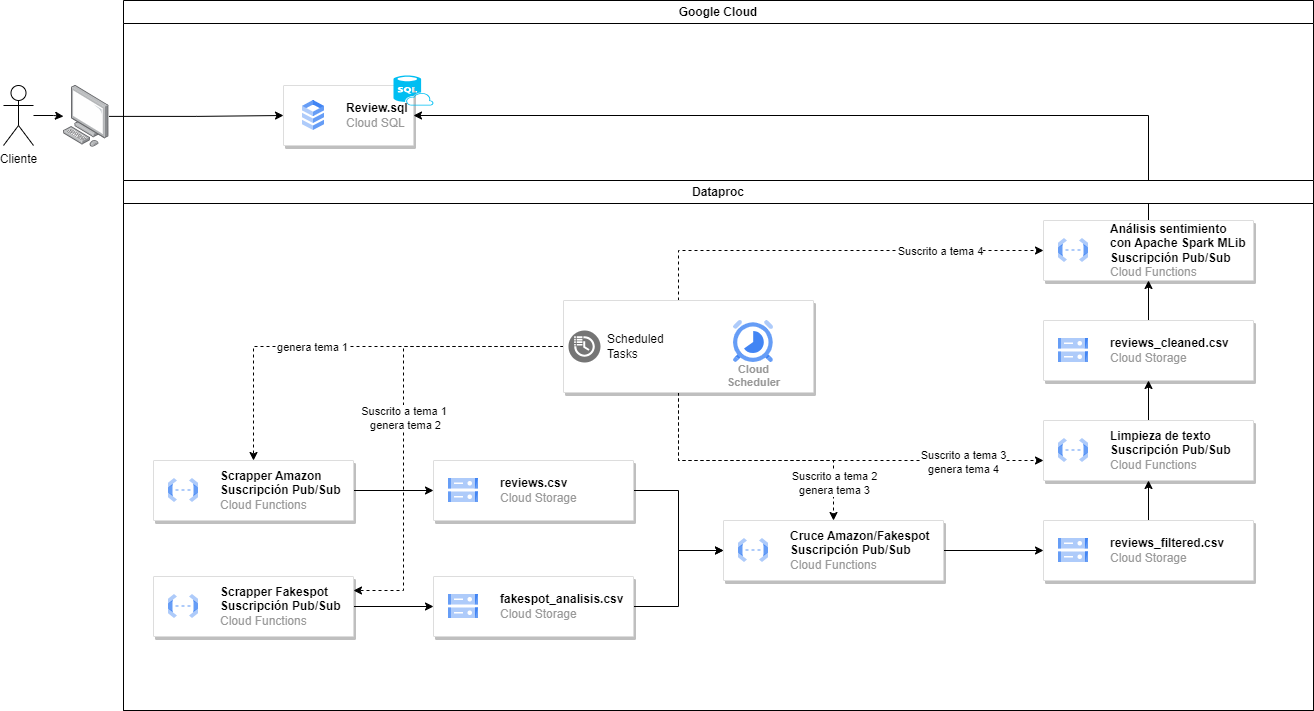
1. Cargar el CSV resultado del Scrapper en **Cloud Storage**
2. Crear un **Cloud Function** con el script para **cruzar los 2 ficheros** y quedarnos con las opiniones que no sean falsas(campo rating != ‘F’)y añadir el código **Pub/Sub** para organizar el flujo de trabajo y que las cloud functions se ejecuten de manera secuencial y programar mediante Scheduler *([ejemplo left join](#bqlbbr5n0a0r) en apartado Desarrollo de la plataforma DAaaS).*
3. Cargar el CSV resultado del cruce de ambos ficheros en **Cloud Storage**
4. Crear una **tabla en Google SQL** para almacenar el resultado del análisis de sentimiento *([ejemplo creación tabla](#a52f1j2bkbbv) en apartado Desarrollo de la plataforma DAaaS).*
5. Crear una **Cloud Function** con el código para la **limpieza del texto** y el código Pub/Sub para que se ejecute después del scrapper y cree otro evento **Pub/Sub** y programar mediante Scheduler seleccionando en el campo evento del scheduler el tema Pub/Sub del Scrapper*([ejemplo limpieza texto](#n26ri5bc49r0) en apartado Desarrollo de la plataforma DAaaS).*:

* Eliminar signos de puntuación
* Eliminar caracteres no alfabéticos
* Eliminar palabras vacías ( ‘el’,’la’,’y’, etc)
* Normalizar el texto
* Dividir el texto en tokens
* Lematizar el texto ( haré, hice,... lo convierte en hacer)

1. Guardar el fichero con el texto limpio en **Google Storage**
2. Crear una **Cloud Function** con el código de **Spark MLlib** usando por ejemplo el algoritmo de clasificación Naive Bayes, y añadiendo el código **Pub/Sub** para poder seleccionar en el Scheduler que se ejecute después del que se genere el tema Pub/Sub de la limpieza de texto *([ejemplo función](#96bm7b1urz5k) en apartado Desarrollo de la plataforma DAaaS).*
3. Guardar el resultado del análisis en la tabla creada en **Google SQL**
4. Programar **Scheduler** para cada cloud function 1 vez al día y en eventos vincularlas a los temas a los que están suscritos para que se ejecuten secuencialmente

### 

Link a Diagrama:

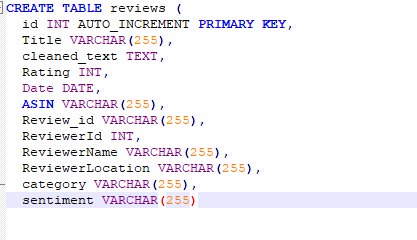


<https://github.com/caroltgn/big-data-architecture.git>

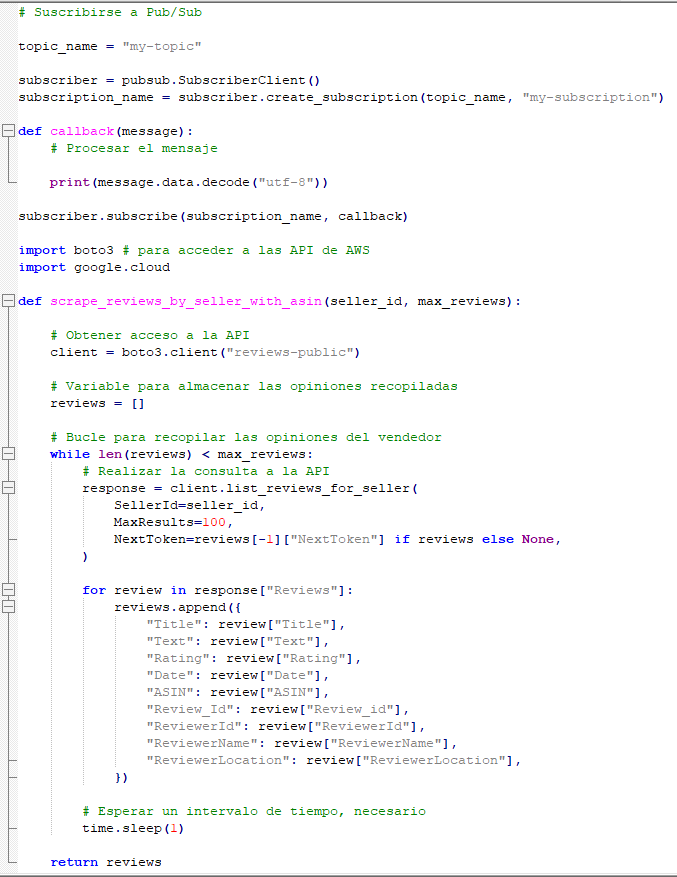
### 

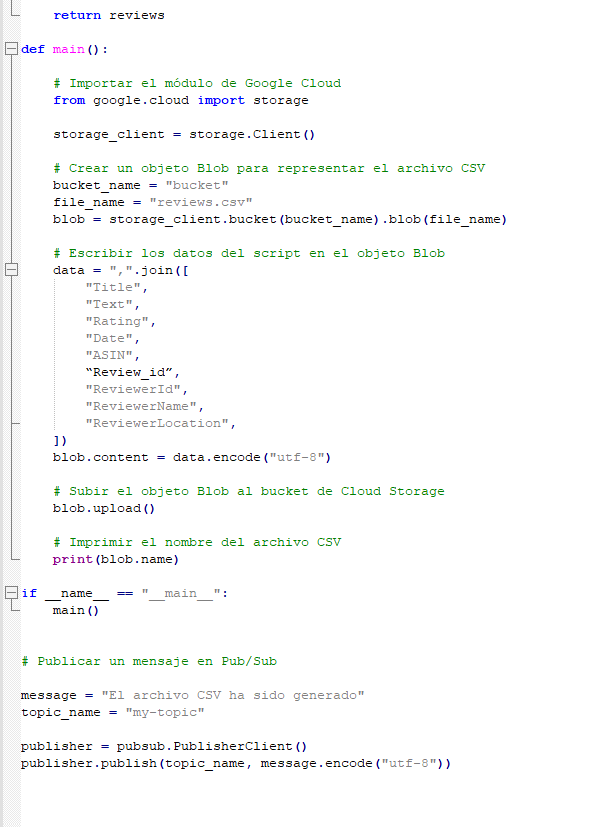
### **Desarrollo de la plataforma DAaaS. (ligera descripción del desarrollo)**

**Ejemplo código para crear la tabla SQL destino:**

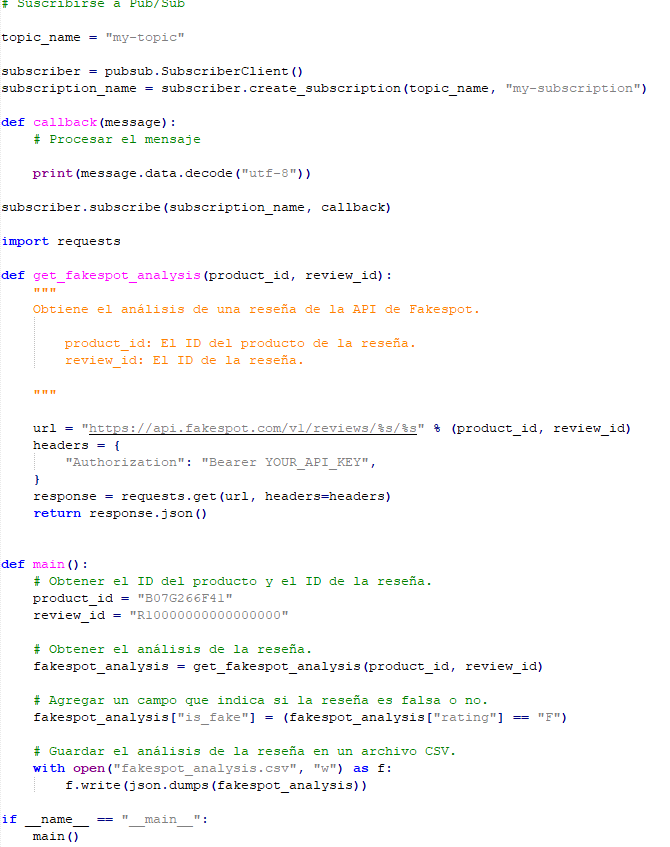
****

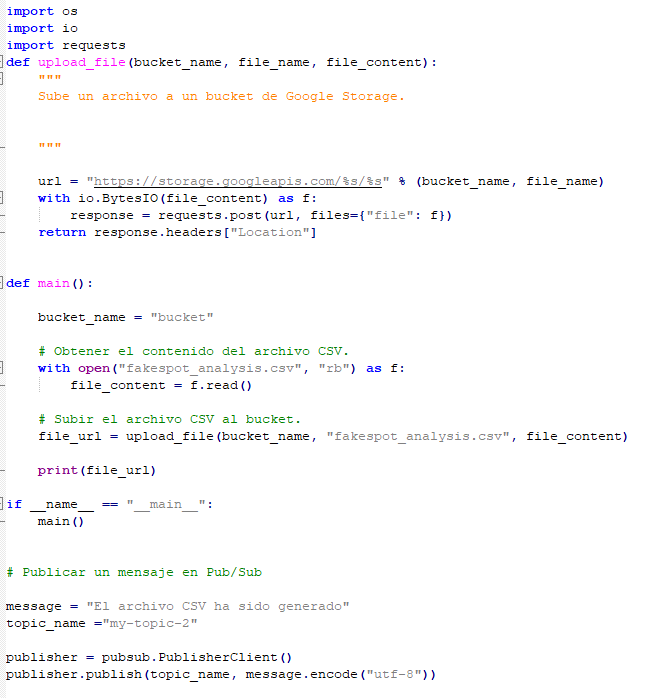
**Ejemplo de código para hacer Scrapper Amazone y que lo guarde en Google Storage:**



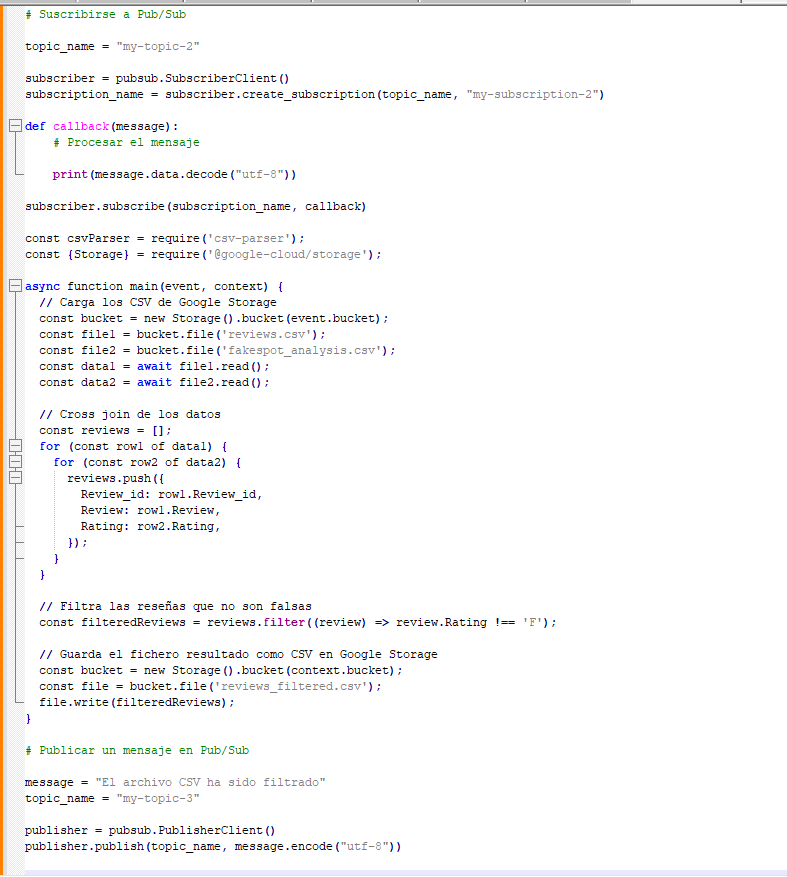


**Scraper de Fakespot que guarda en goggle storage:**

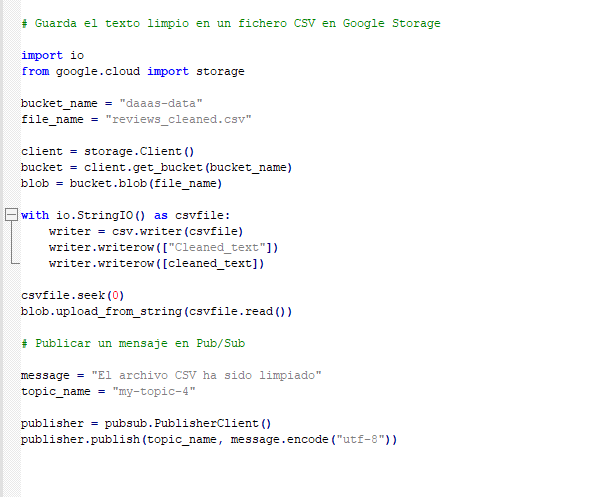
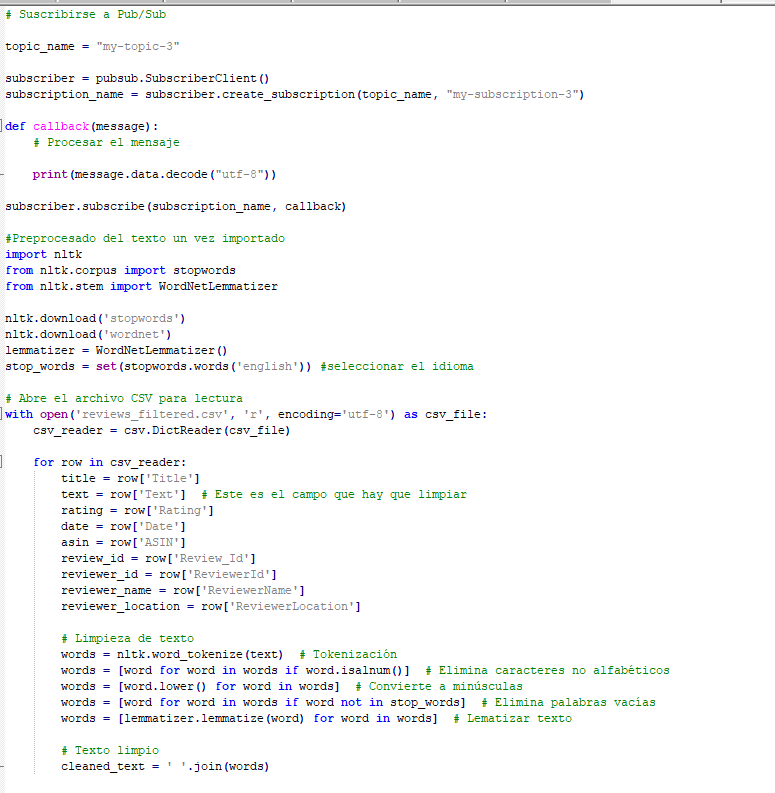
****

****

**Ejemplo de cómo cruzar los 2 ficheros para quedarnos con las opiniones no falsas:**



**Ejemplo de código para preparar el texto:**



**Ejemplo de código para Apache Spark MLlib:**

|  |
| --- |
| # Suscribirse a Pub/Sub  topic\_name = "my-topic-4"  subscriber = pubsub.SubscriberClient()  subscription\_name = subscriber.create\_subscription(topic\_name, "my-subscription-4")  def callback(message):  # Procesar el mensaje  print(message.data.decode("utf-8"))  subscriber.subscribe(subscription\_name, callback)  import sparkmllib.classification.NaiveBayes  *# Crear la sesión de Spark*  spark = SparkSession.builder.appName("Sentiment Analysis").getOrCreate()  *# leer fichero*  data = spark.read.csv("gs://bucket/*reviews\_cleaned.csv*")  *# Agregar la categoría al dataframe*  data = data.withColumn("category", data["cleaned\_text"].cast("string").rlike("price"))  data = data.withColumn("category", data["category"].when(data["category"] == "price", "price").otherwise("quality"))  data = data.withColumn("category", data["category"].when(data["category"] == "quality", "quality").otherwise("customer\_support"))  *# Agregar el sentimiento al dataframe*  data = data.withColumn("sentiment", model.predict(data.select("cleaned\_text")))  # Save the results in append mode  data.write.mode("append").format("org.apache.spark.sql.execution.datasources.jdbc.JdbcDialect").option("url", "jdbc:mysql://my-db:3306/my-database").option("dbtable", "reviews.sql").save() |