**Introducción**

En este pequeño documento abordaremos dos cuestiones. Por un lado, explicaremos brevemente la implementación técnica del script en Python. Por el otro, responderemos las preguntas.

**Resolución del script en Python**

Antes de desarrollar el script, analicé los datos de la tabla de ***data\_engineers*** para detectar diferentes anomalías. En ese sentido, encontré que:

* Existían caracteres extraños, con los cuales había que tener cuidado y chequear que no se pierdan en el pasaje de formato *utf-16-le* al formato *utf-8*;
* Espacios en blanco: Se eliminaron los espacios en blanco de más en los campos, como en el *id=982*, y aquellos que representaban valores nulos se mantuvieron, como el caso del *id=978*, al cual le faltaban los campos *first\_name* y *last\_name*.
* Existían filas que estaban cortadas y su información continuaba en las filas siguientes: Para resolver este inconveniente utilicé los diccionarios de Python, de modo tal que los campos de un registro que están en las filas siguientes los almacena como valores de la clave, que en este caso es el ***id***.

Luego de estas validaciones, chequea que la cantidad total de campos no supere los que son definidos en el encabezado. Es así como, por ejemplo, el *id=29* es descartado por tener 6 campos en vez de 5, mientras que el *id=82* no.

Para la ejecución del script escribí un pequeño proceso *bash*, a los cuales hay que pasarles como argumentos los archivos de entrada (***tsv***) y salida (***csv***).

**Preguntas**

*1. ¿En qué requerimiento implementarías una cola de mensajes en una solución orientada a datos? ¿Qué lenguaje utilizarías y por qué?*

Implementaría una cola de mensajes en el caso de tener que procesar datos provenientes de aplicaciones distribuidas que tengan diferentes componentes ejecutándose en distintos entornos. Este tipo de arquitectura nos permite desacoplar las funcionalidades y lograr una mejor escalabilidad, estabilidad y mantenimiento. Tanto los componentes productores de mensajes como los componentes consumidores de mensajes interactúan con la cola y no entre sí, lo que facilita la comunicación asincrónica.

Existen diferentes arquitecturas de colas de mensajes. Por ejemplo, puedo tener varios productores enviando mensajes a la cola y diferentes consumidores que procesan cada mensaje una sola vez, en un esquema uno a uno; o puedo tener varios productores y varios consumidores que procesan cada mensaje más de una vez, en cuyo caso el esquema sería de publicación-suscripción.

Para llevar a cabo esta tarea puedo utilizar AWS SQS, un servicio que proporciona una API genérica a la cual conectarse con distintos lenguajes, como Java, Node.js, Go o Python. Elegiría utilizar Python, ya que puedo hacer uso de la librería ***boto3*,** que contiene sencillas funciones (*send\_message, receive\_message, delete\_message*, etc) para trabajar con colas.

*2. ¿Qué experiencia posees sobre PySpark o Spark Scala? Contar breves experiencias, en caso de contar experiencia con otro framework de procesamiento distribuido, dar detalles también.*

No desarrollé proyectos con PySpark o Spark Scala pero leí algo de código y documentación sobre PySpark. Entiendo que es un lenguaje que facilita escalar en el análisis, el desarrollo de ETLs y la construcción de modelos de machine learning sobre estructuras de datos distribuidas en clusters de Spark. Además, a diferencia de otros frameworks para procesamiento distribuido como Hadoop, facilita el desarrollo de procesos en tiempo real. Los dataframes de Spark tienen funcionalidades similares a las de Pandas o R, lenguaje con el cual desarrollé varios proyectos. Una de las ventajas de PySpark sobre librerías como Pandas es que la ejecución de la carga y las transformaciones de los dataframes se demoran hasta que sean necesarias, permitiéndonos ahorrar recursos. También podemos realizar consultas sobre dataframes de Spark mediante SparkSQL, y si nos manejamos con Databricks podemos crear algunas visualizaciones en nuestro entorno local.

*3. ¿Qué funcionalidad podrías expandir desde el área de ingeniería de datos con una API y arquitectónicamente cómo lo modelarías?*

Una API nos da una serie de ventajas. Por un lado, nos permite una mayor automatización, tanto para la subida como para la bajada de información, y una reducción de la cantidad de tareas que tiene que llevar a cabo el desarrollador. Por otro lado, nos brinda mayor versatilidad para acceder a los conjuntos de datos necesarios. Por ejemplo, podemos pasarle parámetros al *request* y obtener diferentes columnas, aplicar filtros o joinear tablas, entre otras cosas. Además, una API nos permite integrar los servicios de alguna plataforma web en múltiples sistemas y con diferentes lenguajes.

En el caso del script que desarrollé para transformar el archivo de ***data\_engineers.tsv***, haría una arquitectura con esta forma:

A close up of a logo

Description automatically generated

Como ejemplo de entidades agregué ***Recursos Humanos*** y ***Project Managers***, suponiendo que la estructura de datos tiene componentes relacionados a los diferentes roles dentro de una organización.

En líneas generales, el cliente que desarrollemos se comunicará con la API para extraer o cargar datos en algún formato específico, como puede ser *json*. En cuanto a la implementación, podemos desarrollar funciones que transformen esos datos en algún otro formato de acuerdo a una lógica específica.