**Solución Propuesta: Procesamiento Distribuido con Apache Spark**

**1. Creación del RDD**

Para comenzar, se utilizaría la función sc.textFile("ruta/del/archivo") para crear un **RDD** a partir de los archivos de datos recibidos. Esta función permite cargar archivos desde HDFS, S3, o el sistema de archivos local, distribuyendo automáticamente los datos entre los nodos del clúster.

*rdd = sc.textFile("s3://datastream/input/datos.csv")*

**2. Transformaciones sobre el RDD**

A continuación, se aplicarían las siguientes **transformaciones** para preparar los datos:

* **Filtrado de registros inválidos**:

*rdd\_filtrado = rdd.filter(lambda linea: len(linea.split(",")) == 5)*

* **Combinación de datos (por ejemplo, unir con otro RDD de referencia):**

*rdd\_combinado = rdd\_limpio.map(lambda campos: (campos[0], campos)).join(rdd\_referencia*

**3. Acciones para obtener resultados**

Para generar los reportes solicitados, se utilizarían las siguientes **acciones**:

* **count()**: para obtener el número total de registros válidos.

*total\_registros = rdd\_limpio.count()*

**collect()** o **saveAsTextFile()**: para recolectar los resultados o guardarlos en un archivo.

*rdd\_resultado.saveAsTextFile("s3://datastream/output/reporte\_final")*

**4. ¿Qué es un Job Spark?**

Un **Job Spark** es una unidad de trabajo que se ejecuta cuando se llama a una acción sobre un RDD. En este contexto, cada vez que se ejecuta una acción como count() o saveAsTextFile(), Spark genera un Job que se divide en etapas (stages) y tareas (tasks), distribuidas entre los nodos del clúster para su ejecución paralela.

**5. Reflexión**

**¿Por qué es importante el procesamiento distribuido?**

El procesamiento distribuido permite manejar grandes volúmenes de datos de forma eficiente, aprovechando múltiples nodos para paralelizar tareas. Esto reduce significativamente el tiempo de ejecución y mejora la escalabilidad del sistema.

**¿Qué desafíos técnicos podrían surgir?**

* **Gestión de recursos**: asegurar que el clúster tenga suficiente capacidad para procesar los datos.
* **Fallos de nodos**: manejar la tolerancia a fallos mediante replicación y reintentos.
* **Optimización de rendimiento**: ajustar particiones, caché y persistencia para mejorar la eficiencia.
* **Calidad de datos**: lidiar con datos corruptos, incompletos o mal formateados.

Esquema

