Propuesta de Arquitectura en la Nube

- **Proponga una arquitectura basada en la nube para desplegar un modelo/proceso de NLP en Batch (Utilice preferentemente la nube de AWS). El modelo debe permitir realizar tareas de clasificación o extracción de información a partir de textos.**

Arquitectura propuesta en AWS

1. **Almacenamiento de Datos de Entrada - Amazon S3**: Es utilizado para almacenar tanto los datos de entrada como los resultados de salida. Los archivos de texto o documentos se suben al bucket de S3 para su procesamiento, y este bucket también actúa como repositorio de los resultados de inferencia.

2. **Notificación de Nuevos Datos - Amazon S3 Event Notification:** Se configura una notificación de eventos en el bucket de S3, de modo que, al subir nuevos archivos de texto o reseñas, se envíe un mensaje a una cola (como Amazon SQS) o se active directamente una función de AWS Lambda. Esto garantiza que los datos sean procesados inmediatamente después de estar disponibles en S3.

3. **Preparación y Transformación de Datos - AWS Glue:** AWS Glue se emplea para realizar las tareas de ETL (Extracción, Transformación y Carga) en los datos de entrada. Limpia y transforma los datos (por ejemplo, eliminando caracteres especiales o realizando tokenización) para prepararlos para el modelo de NLP. Los datos transformados se almacenan en un bucket de S3 específico, desde donde AWS Batch los carga para la inferencia.

4. **Procesamiento Batch - AWS Batch:** AWS Batch ejecuta el procesamiento en lotes de los archivos de texto, escalando automáticamente según el volumen de datos y el tamaño del lote. AWS Batch lanza contenedores (usando imágenes en Amazon ECR o modelos en SageMaker) para realizar las inferencias en batch sobre los archivos de entrada.

5. **Gestión del Modelo - Amazon SageMaker o Amazon ECR:** Amazon SageMaker o Amazon ECR se utiliza para almacenar y gestionar el modelo de NLP. SageMaker permite el entrenamiento y despliegue de modelos NLP, aprovechando frameworks como PyTorch, TensorFlow y Hugging Face Transformers. Para modelos preentrenados o empaquetados en contenedores de Docker, Amazon ECR es la opción ideal de almacenamiento.

6. **Almacenamiento de Resultados - S3:** Los resultados, como etiquetas de clasificación o información extraída, se guardan en un bucket específico en Amazon S3. Opcionalmente, los resultados también pueden almacenarse en una base de datos como Amazon RDS o Amazon DynamoDB para consultas rápidas o almacenamiento estructurado.

7. **Monitoreo y Logging - Amazon CloudWatch:** Amazon CloudWatch monitorea y administra los trabajos en AWS Batch y Lambda. Las alarmas de CloudWatch notifican sobre errores, tiempos de ejecución largos o cualquier anomalía que requiera atención.

8. **Notificaciones de Procesamiento - Amazon SNS (Simple Notification Service):** Al completarse el procesamiento en batch, Amazon SNS envía una notificación para informar sobre la finalización del proceso, ya sea por correo electrónico u otro canal de notificación preferido.

Diagrama Resumido de la Arquitectura

1. **Entrada de Datos** -> Amazon S3 (Bucket de Datos de Entrada)
2. **Notificación de Nuevos Datos** -> S3 Event Notification + Amazon SQS
3. **Preparación de Datos** -> AWS Glue (ETL y transformación de datos)
4. **Gestión de Trabajo** -> AWS Batch (Instancias EC2 o ECS)
5. **Procesamiento NLP** -> Amazon SageMaker / Contenedor Docker en AWS Batch
6. **Almacenamiento de Resultados** -> Amazon S3 (Bucket de Resultados)
7. **Monitoreo y Logging** -> Amazon CloudWatch y AWS CloudTrail
8. **Notificación Final** -> Amazon SNS

- **Explique cómo la arquitectura facilita escalabilidad y confiabilidad frente a grandes volúmenes de datos.**

1. **Escalabilidad**
   La arquitectura facilita la escalabilidad mediante la capacidad de ajustar dinámicamente los recursos de cómputo y almacenamiento según la carga de trabajo y la cantidad de datos, sin necesidad de intervención manual.

   - **AWS S3 como Almacenamiento Escalable:** Amazon S3, un sistema de almacenamiento de objetos que se escala automáticamente, maneja grandes volúmenes de datos sin requerir administración compleja. Al almacenar los datos de entrada y los resultados en S3, la arquitectura evita cuellos de botella, permitiendo que millones de archivos y solicitudes sean procesados de manera simultánea y eficiente.

   - **AWS Batch para Procesamiento Dinámico y Escalable:** AWS Batch gestiona cargas de trabajo en batch ajustando el número de instancias según los requisitos de cada lote de datos. Este servicio lanza instancias de EC2 o contenedores ECS de forma dinámica, basándose en el volumen de datos y la complejidad del modelo NLP. Por ejemplo, al recibir un millón de reseñas para procesar, AWS Batch divide el trabajo en múltiples tareas y asigna los recursos necesarios para procesar cada una de forma paralela, reduciendo significativamente el tiempo total de procesamiento.

   - **Amazon SageMaker y Contenedores en AWS Batch:** Tanto Amazon SageMaker como los contenedores Docker en AWS Batch permiten empaquetar el modelo NLP de manera modular y reutilizable. Esto facilita la adición o actualización de modelos sin interrumpir el flujo de

procesamiento. SageMaker, además, se ajusta automáticamente a la demanda de recursos. En caso de que se necesiten varios modelos para distintas tareas (como clasificación y extracción de información), cada modelo puede ejecutarse en su propio contenedor, escalando independientemente según la demanda.

2. **Confiabilidad**
   La arquitectura está diseñada para ser altamente confiable y resistente frente a fallos o problemas, lo cual es crucial para sistemas en producción y para el procesamiento de grandes volúmenes de datos, donde los errores pueden tener un impacto significativo:

- **Tolerancia a Fallos con Amazon SQS y AWS Lambda:** Al usar Amazon SQS para gestionar las notificaciones de nuevos datos, se asegura que las tareas en batch solo se activan cuando los datos están disponibles, y el sistema puede retener estos mensajes hasta que los trabajos se procesen correctamente. Si un job falla, AWS Batch puede reintentar automáticamente la tarea, ejecutándola en una nueva instancia o en un contenedor de reemplazo, aumentando la confiabilidad del procesamiento en batch.

- **Monitorización y Alarmas con Amazon CloudWatch:** Amazon CloudWatch permite monitorear cada job y configurar alarmas para alertar sobre errores, sobrecarga o problemas de rendimiento. Esto garantiza que los administradores del sistema puedan actuar de inmediato en caso de problemas.
  Por ejemplo, si un job se ejecuta más tiempo de lo esperado o recibe una cantidad inusual de datos, CloudWatch puede alertar al equipo, y, si es necesario, ajustar la capacidad de procesamiento.

- **Distribución del Trabajo y Redundancia:** Al dividir el procesamiento en múltiples trabajos independientes mediante AWS Batch, la arquitectura evita puntos únicos de falla. Si una instancia o contenedor falla, otros trabajos en paralelo continúan sin interrupciones, asegurando que el sistema siga funcionando y complete el procesamiento general, incluso frente a fallas aisladas.

- **Amazon S3 para Almacenamiento Resiliente de Datos y Resultados:** Amazon S3 proporciona una durabilidad de 99.999999999% (11 nueves) en los objetos almacenados, lo que protege y asegura la disponibilidad de los datos de entrada y los resultados, incluso en caso de fallas o errores de red. S3 también permite replicación en múltiples regiones y configuraciones de versionado, añadiendo seguridad y disponibilidad a los datos.

- **Automatización de Procesos para Reducir el Error Humano:** La arquitectura aprovecha funciones como notificaciones automáticas, procesamiento batch y almacenamiento centralizado en S3, minimizando la intervención humana. Esto no solo optimiza el procesamiento, sino que también reduce la probabilidad de errores humanos, mejorando así la confiabilidad del sistema.

- **Explique la elección de los componentes para la preparación de datos, trabajos ETL, implementación de modelos y su papel en la arquitectura.**

1. **Preparación de Datos**
   Para la preparación de datos, los componentes seleccionados son Amazon S3 y AWS Glue:

   - **Amazon S3 (Simple Storage Service):** La flexibilidad de S3 permite organizar y versionar datos y almacenar tanto datos estructurados como no estructurados (e.g., archivos JSON, CSV o Parquet). Su escalabilidad y alta disponibilidad facilitan el almacenamiento de grandes volúmenes de datos de entrada y aseguran que los datos estén accesibles para otros servicios de AWS, como Glue, Batch y SageMaker.

   - **AWS Glue para ETL y Preparación de Datos:** AWS Glue es un servicio ETL totalmente administrado, ideal para extraer, transformar y cargar datos desde múltiples fuentes. En esta arquitectura, Glue realiza tareas como limpieza de texto, eliminación de caracteres especiales, tokenización y extracción de características, preparando así los datos de entrada para el modelo NLP.

2. **Trabajos ETL (Extracción, Transformación y Carga)**
   Los trabajos ETL son fundamentales para garantizar que los datos estén en el formato adecuado antes de ser procesados por el modelo. Los componentes clave aquí son AWS Glue y AWS Lambda:

   - **AWS Glue:** AWS Glue realiza el procesamiento ETL a gran escala. Gracias a sus crawlers y motores de transformación, Glue puede explorar datos, inferir esquemas automáticamente y generar transformaciones personalizadas en PySpark. Una vez que los datos se han limpiado y transformado, Glue los guarda en S3 en un formato adecuado (e.g., CSV, Parquet o JSON) para que AWS Batch o SageMaker los procesen fácilmente.

   - **AWS Lambda para Orquestación de ETL:** AWS Lambda ejecuta scripts ligeros que automatizan el flujo de datos y manejan eventos. Por ejemplo, Lambda puede configurarse para activarse cada vez que se suben datos nuevos a S3, desencadenando trabajos ETL en AWS Glue o iniciando tareas en AWS Batch. Lambda actúa como orquestador de eventos, asegurando que el procesamiento ETL se ejecute cuando sea necesario, manteniendo la eficiencia y evitando costos adicionales.

3. **Implementación del Modelo NLP**
   Para implementar el modelo NLP, los componentes clave seleccionados son Amazon SageMaker y AWS Batch:

   - **Amazon SageMaker para Entrenamiento e Implementación del Modelo:** SageMaker facilita el entrenamiento y despliegue de modelos NLP, permitiendo utilizar frameworks populares como TensorFlow, PyTorch y Hugging Face Transformers. Una vez entrenado el modelo, SageMaker permite exportarlo para trabajos en batch o desplegarlo a través de endpoints.

- **AWS Batch para Procesamiento Batch de NLP:** AWS Batch es ideal para la inferencia en batch del modelo NLP. En lugar de procesar cada reseña en tiempo real, AWS Batch agrupa los datos y realiza inferencias en lotes grandes, optimizando los recursos y reduciendo costos. Batch puede lanzar instancias EC2 o contenedores ECS que procesan los datos almacenados en S3, ejecutan inferencias y guardan los resultados de vuelta en S3. Esta estructura permite procesar grandes volúmenes de datos en paralelo, mejorando la eficiencia.

4. **Monitoreo y Gestión de la Arquitectura**

   - **Amazon CloudWatch para Monitoreo:** CloudWatch permite monitorear los trabajos de Glue, AWS Batch y los endpoints de SageMaker. Con métricas personalizadas y alarmas, los equipos pueden identificar problemas de rendimiento y fallos en el procesamiento. Esto asegura una operación confiable y permite intervenir rápidamente en caso de errores. CloudWatch también registra logs de cada etapa del proceso ETL y de inferencia en batch, proporcionando visibilidad completa del flujo de trabajo.

   - **Amazon SQS para Notificaciones:** SQS permite enviar alertas automáticas a administradores o equipos de desarrollo ante eventos específicos, como la finalización de un trabajo batch o un error crítico. Esto facilita una respuesta rápida ante problemas operativos y asegura la continuidad del sistema.

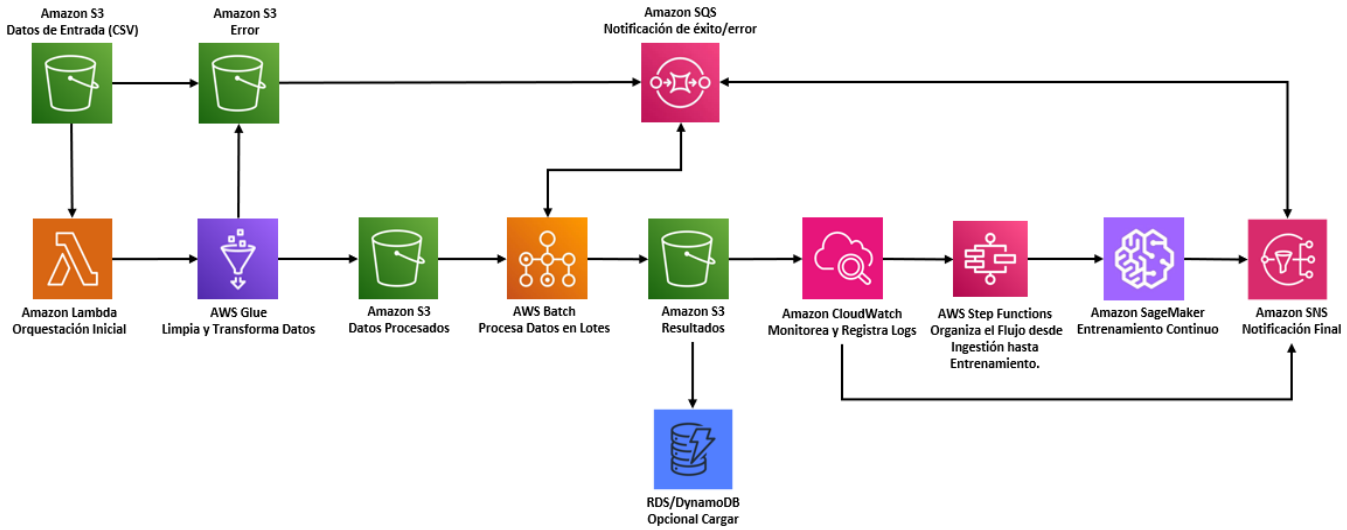
Resumen del Rol de Cada Componente

1. **Amazon S3:** Almacenamiento de datos brutos, datos transformados y resultados de inferencia; es el punto central de almacenamiento de datos en la arquitectura.
2. **AWS Glue:** Realiza tareas ETL para preparar y transformar los datos, facilitando el procesamiento en batch y asegurando que los datos están en el formato correcto para el modelo.
3. **AWS Lambda:** Automatiza los flujos de trabajo y desencadena procesos en respuesta a eventos, como la llegada de nuevos datos.
4. **Amazon SageMaker:** Entorno de entrenamiento e implementación de modelos NLP, facilita el ajuste de modelos y la exportación para inferencias en batch.
5. **AWS Batch:** Ejecuta trabajos de inferencia en batch, procesando grandes volúmenes de datos en paralelo y utilizando el modelo NLP preentrenado.
6. **CloudWatch y SQS:** Monitoreo, logging y alertas para asegurar confiabilidad y visibilidad de todo el proceso.

Step-by-Step

- **Diagrame la secuencia de pasos desde la ingestión de datos hasta el monitoreo y el entrenamiento continuo, incluyendo la orquestación.**

A continuación, se muestra el diagrama propuesto:



- **Explique cómo contribuye el componente de orquestación a la ejecución del pipeline de NLP.**

  El componente de orquestación AWS Step Functions desempeña un rol clave en la ejecución del pipeline de NLP al:

  - **Coordinar cada etapa:** Asegura que los servicios (S3, Lambda, Glue, Batch, SageMaker) se ejecuten en el orden correcto, evitando errores de sincronización.
  - **Gestionar errores y reintentos:** Reintenta automáticamente las tareas en caso de fallos y envía notificaciones de error mediante Amazon SQS o SNS, mejorando la confiabilidad del pipeline.
  - **Ejecutar condiciones dinámicas:** Permite rutas condicionales en el flujo, adaptando el pipeline a diferentes tipos de datos.
  - **Automatizar el entrenamiento continuo:** Activa Amazon SageMaker para reentrenar el modelo de NLP cuando se acumulan suficientes datos, manteniendo el modelo actualizado.
  - **Monitoreo y notificaciones:** Registra logs en CloudWatch y envía notificaciones al finalizar, proporcionando visibilidad del rendimiento y alertando en caso de incidentes.

En conjunto, Step Functions hace que el pipeline sea escalable, confiable y fácil de mantener, permitiendo el procesamiento en batch de NLP con alta eficiencia y automatización.

Estructura de Directorios

- **Proponga una estructura de directorios para el proyecto que mejore la organización y mantenimiento del código.**

```
project-root/
│
├── README.md                 # Descripción del proyecto y guía de inicio
├── requirements.txt          # Lista de dependencias de Python
├── setup.py                  # Configuración para instalar paquetes del proyecto
├── .gitignore                # Archivos y carpetas a ignorar en Git
│
├── data/                     # Carpeta para datos de entrada y salida (solo para desarrollo local)
│   ├── raw/                  # Datos de entrada originales
│   ├── processed/            # Datos procesados y listos para el modelo
│   └── results/              # Resultados de las pruebas o inferencias locales
│
├── src/                      # Código fuente del proyecto
│   ├── __init__.py           # Marca la carpeta como un paquete Python
│   ├── data_processing/      # Módulos para preparación y transformación de datos
│   │   ├── __init__.py
│   │   ├── preprocess.py     # Script de limpieza y preprocesamiento de datos
│   │   └── feature_engineering.py  # Script para creación de features o embeddings
│   │
│   ├── model/                # Código relacionado con el modelo de NLP
│   │   ├── __init__.py
│   │   ├── train.py          # Script para el entrenamiento del modelo
│   │   ├── evaluate.py       # Script para evaluar el modelo
│   │   └── infer.py          # Script para la inferencia de textos nuevos
│   │
│   ├── batch_processing/     # Scripts para procesamiento en batch
│   │   ├── __init__.py
│   │   ├── batch_job.py      # Script principal de procesamiento en batch
│   │   └── aws_batch_handler.py  # Módulo para configurar y ejecutar trabajos en AWS Batch
│   │
│   ├── utils/                # Utilidades y funciones auxiliares
│   │   ├── __init__.py
│   │   ├── s3_utils.py       # Funciones para interactuar con S3
│   │   ├── glue_utils.py     # Funciones para interactuar con AWS Glue
│   │   └── logging_utils.py  # Funciones de logging y manejo de errores
│
├── configs/                  # Archivos de configuración
│   ├── config.yaml           # Configuración general del proyecto
│   ├── model_config.yaml     # Configuración del modelo (hiperparámetros, arquitectura)
│   └── aws_config.yaml       # Configuración específica de AWS (ARNs, buckets, roles)
│
├── scripts/                  # Scripts para tareas administrativas y de infraestructura
│   ├── deploy.sh             # Script de despliegue en AWS (Batch, Lambda, Glue)
│   ├── build_docker.sh       # Script para construir y subir imágenes Docker a ECR
│   ├── create_s3_buckets.py  # Script para crear buckets de S3
│   └── setup_iam_roles.py    # Script para configurar roles y permisos de IAM
│
├── docker/                   # Configuración para contenedores Docker
│   ├── Dockerfile            # Dockerfile para construir la imagen del modelo
│   └── batch_Dockerfile      # Dockerfile específico para trabajos en AWS Batch
│
└── docs/                     # Documentación del proyecto
    ├── architecture_diagram.png  # Diagrama de arquitectura
    ├── workflow_overview.md      # Explicación del flujo de trabajo
    └── setup_guide.md            # Guía de configuración y despliegue en AWS
```

Descripción de cada carpeta:

- **data/:** Carpeta para los datos en local, útil en la fase de desarrollo y prueba. En producción, los datos serán almacenados en S3. Incluye subcarpetas para datos en crudo, datos procesados y resultados.
- **src/:** Contiene el código fuente del proyecto, dividido en submódulos para facilitar el mantenimiento y la colaboración.
- **data_processing/:** Scripts de preprocesamiento y transformación de datos.
- **model/:** Código para entrenamiento, evaluación e inferencia del modelo de NLP.
- **batch_processing/:** Scripts para configurar y ejecutar trabajos en batch, incluyendo los scripts para AWS Batch.
- **utils/:** Funciones auxiliares que se pueden reutilizar en diferentes partes del proyecto (como manejo de S3, Glue, y logs).
- **configs/:** Archivos de configuración en formato YAML. Organizados para separar la configuración general, la específica del modelo, y la configuración de AWS (ARNs, buckets, roles).
- **scripts/:** Scripts de administración e infraestructura. Aquí se incluyen los scripts para desplegar en AWS, crear roles y permisos de IAM, construir imágenes Docker, y otros.
- **docker/:** Archivos relacionados con Docker, incluyendo Dockerfiles para construir las imágenes del modelo y las imágenes específicas para trabajos en AWS Batch.
- **docs/:** Documentación del proyecto. Aquí se pueden incluir diagramas de arquitectura, explicaciones del flujo de trabajo, y una guía de configuración y despliegue en AWS.

- **¿Cómo manejaría la versión del pipeline de preprocesamiento y los modelos entrenados en el directorio de modelos?**

```
project-root/
├── preprocessing_pipelines/
│   ├── pipeline_v1.py
│   ├── pipeline_v2.py
│   └── latest -> pipeline_v2.py     # Enlace simbólico a la última versión
│
└── models/
    ├── model_v1/
    │   ├── model.pkl
    │   ├── preprocessor.pkl
    │   ├── config.yaml
    │   └── metrics.json
    │
    ├── model_v2/
    │   ├── model.pkl
    │   ├── preprocessor.pkl
    │   ├── config.yaml
    │   └── metrics.json
    │
    └── latest/              # Enlace simbólico a la versión más reciente
```

Estrategia de Versionado

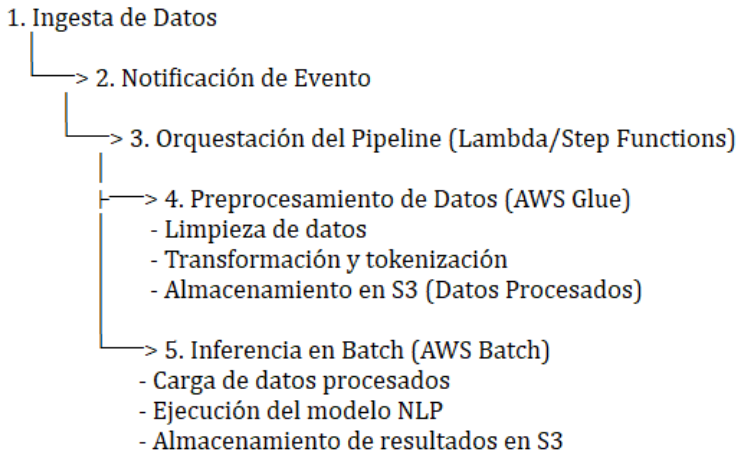
1. **Pipelines:** Guardar cada versión del pipeline (pipeline_v1.py, pipeline_v2.py) y utilizar un enlace simbólico latest apuntando a la última versión.

2. **Modelos**: Cada versión del modelo tiene su propia carpeta (model_v1, model_v2) que incluye:
   - Archivo del modelo (model.pkl).
   - Pipeline de preprocesamiento (preprocessor.pkl).
   - Configuración (config.yaml).
   - Métricas (metrics.json).

3. **Última Versión:** Usar un enlace simbólico latest en cada carpeta para apuntar a la versión en producción, permitiendo un fácil acceso a la versión actual.

Pipeline de procesamiento de datos

- **Diagrame y explique la secuencia de pasos desde la ingesta de datos hasta la inferencia en batch.**



  - **Ingesta de Datos (S3):** Los datos en bruto, como archivos de texto o documentos de reseñas, se cargan en un bucket de Amazon S3. Este bucket de entrada almacena todos los datos que necesitan ser procesados y es el punto de partida del pipeline.
  - **Notificación de Evento (S3 Event Notification):** Cuando se carga un nuevo archivo en el bucket de S3, se activa una notificación de evento que desencadena el flujo de trabajo. Esta notificación se configura para enviar un mensaje a AWS Lambda o Amazon SQS, activando automáticamente el pipeline sin intervención manual.
  - **Orquestación del Pipeline (AWS Lambda / Step Functions):** AWS Lambda o AWS Step Functions actúa como el orquestador principal del pipeline, activando cada etapa en el orden adecuado. Una vez que recibe la notificación de nuevos datos, Lambda o Step Functions

coordina el flujo de procesamiento, primero llamando a AWS Glue para realizar el preprocesamiento de datos y luego ejecutando el job en AWS Batch para la inferencia.

- **Preprocesamiento de Datos (AWS Glue):** AWS Glue realiza el procesamiento inicial de los datos, limpieza de datos, transformación y Tokenización y luego, los datos transformados se almacenan en un bucket de S3.
- **Inferencia en Batch (AWS Batch):** AWS Batch carga los datos procesados desde S3 y ejecuta el modelo de NLP en modo batch. Durante esta etapa toma los datos procesados almacenados en S3, ejecuta el modelo NLP realizando tareas de inferencia, como clasificación o extracción de información, sobre los datos de entrada en lotes. Por último, los resultados de la inferencia, como etiquetas de clasificación o entidades extraídas, se guardan en otro bucket de S3.

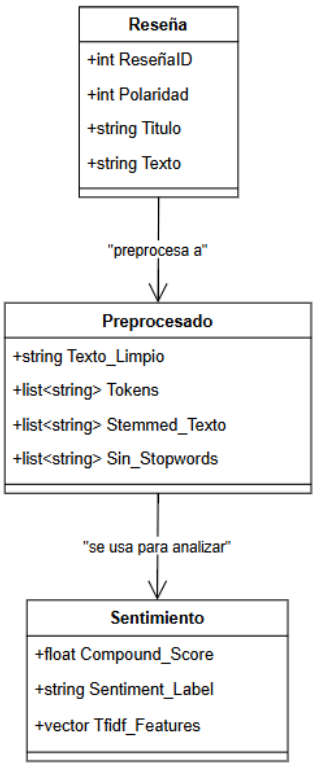
- **Explore los datos para hacer un paso a paso de la limpieza y calidad que tienen, donde muestre qué estrategia utilizó para sacar el máximo provecho.**

  La respuesta a este punto con el paso a paso correspondiente está disponible en el notebook de Python.

- **Incluya un diccionario de datos.**

Nombre de Columna	*Tipo de Dato*	*Descripción*
polaridad	int (0 o 1)	Indica el sentimiento de la reseña: 0 para negativa, 1 para positiva.
titulo	str	Título original de la reseña en Amazon.
texto	str	Cuerpo de la reseña en Amazon, junto al título concatenado (columna generada en el preprocesamiento).
texto_limpio	str	Texto después de la limpieza (sin URLs, menciones, ni puntuación) y en minúsculas.
tokens	List o str	Lista de palabras individuales (tokens) generadas a partir de texto_limpio.
sin_stopwords_texto	List o str	Lista de palabras en tokens tras eliminar stopwords y palabras comunes.
stemmed_texto	List o str	Lista de palabras en tokens después de aplicar stemming (reducción a raíces).
compound	float	Puntuación de sentimiento compuesta calculada con VADER Sentiment Analysis, en el rango [-1, 1].
sentiment	str	Clasificación de sentimiento basada en compound: "Positivo", "Negativo" o "Neutral".
tfidf_features (X_train, X_test)	scipy.sparse matrix	Matriz de características generada con TfidfVectorizer, donde cada reseña se convierte en un vector numérico.

- **Trace el modelo de datos conceptual y explique la selección de este.**



Justificación del Modelo Conceptual

- **Modularidad**: Separar los datos en tres entidades distintas permite que cada etapa (ingesta, preprocesamiento y análisis de sentimiento) se realice de manera independiente y optimizada.

- **Estandarización y Consistencia:** Al tener Preprocesado y Sentimiento separados de Reseña, aseguramos que los datos que llegan al modelo están limpios, estandarizados y listos para su uso.

- **Escalabilidad:** Este diseño facilita la incorporación de nuevas técnicas de preprocesamiento o análisis de sentimiento sin modificar los datos originales, manteniendo flexibilidad en el pipeline.

- **Reusabilidad:** Con esta estructura, los datos procesados y las características numéricas pueden reutilizarse para distintas pruebas de modelo o análisis, sin repetir tareas de procesamiento o cálculo.

Pipeline de CI/CD/CT

- **Construya un pipeline de CI/CD/CT (usando GitHub Actions preferiblemente).**

  Descripción del Pipeline Completo

  - **Job test**: Ejecutar Pruebas Unitarias. Este job instala las dependencias y ejecuta las pruebas unitarias en test_pipeline.py para asegurar que el código funcione correctamente.
  - **Job build:** Construir el Proyecto. Este job simula la construcción de la aplicación y solo se ejecuta si el job test pasa.
  - **Job deploy:** Despliegue del Proyecto. Simula el despliegue de la aplicación, dependiendo de que el job build se complete exitosamente.
  - **Job train:** Entrenamiento Continuo del Modelo. Este job ejecuta el script train_model.py para realizar el entrenamiento del modelo.

  En el repositorio en Github se encuentra el archivo creado **.github/workflows/ci-cd-pipeline.yml**

- **Explique cómo se ejecutaría el pipeline de entrenamiento continuo.**

  El pipeline de entrenamiento continuo (CT) se activa automáticamente cuando se hace un push o se crea un pull request en la rama principal, lo que garantiza que el modelo esté siempre actualizado.

  Flujo del Pipeline CT

  - **Configuración del Entorno:** Clona el repositorio, configura Python y las dependencias necesarias desde requirements.txt.
  - **Entrenamiento del Modelo:** Ejecuta train_model.py, que utiliza los datos más recientes para entrenar el modelo y, si está programado, lo evalúa automáticamente para asegurar su rendimiento.
  - **Almacenamiento y Despliegue:** Guarda el modelo entrenado en un repositorio o almacenamiento en la nube, listo para su uso en producción. Puede incluir versiones del modelo para facilitar el seguimiento de mejoras.
  - **Notificación:** El pipeline genera notificaciones en GitHub Actions (y opcionalmente, en otros canales) sobre el estado de éxito o fallo del entrenamiento.

  Este flujo automatiza el reentrenamiento y despliegue del modelo, asegurando que siempre esté adaptado a los cambios en los datos sin intervención manual.

Monitoreo del Pipeline/Modelo

- **Proponer un esquema de monitoreo para el pipeline de procesamiento batch.**

1. **Esquema de Monitoreo del Pipeline Batch**

   - **Monitoreo de Integridad del Pipeline:** Usar CloudWatch para registrar logs y métricas de cada ejecución del pipeline. Monitorear: Duración de los trabajos (AWS Batch GitHub Actions), uso de recursos (CPU, memoria) para identificar cuellos de botella, estado de finalización (exitoso o fallido) de cada paso del pipeline.
   - **Monitoreo de Datos en Tiempo Real:** SageMaker Model Monitor para implementar reglas de validación de datos, como verificación de nulos o desviaciones inesperadas en características clave. Enviar alertas desde Amazon SNS si los datos no cumplen con los criterios establecidos, o registrar un evento en los logs para el seguimiento.

2. **Métricas Relevantes a Monitorear**

   - **Tiempo de Ejecución del Pipeline**: Usar CloudWatch Logs para registrar el tiempo que toma cada paso del pipeline. Un aumento en el tiempo de ejecución puede indicar problemas de rendimiento o datos demasiado grandes para procesarse en el tiempo esperado.
   - **Estado de los Jobs en el Pipeline:** GitHub Actions para rastrear el éxito o fallo de cada job (test, build, deploy, train), con información en tiempo real sobre el estado de cada paso. AWS Step Functions permite monitorear la ejecución y estado de cada componente del pipeline.
   - **Métricas del Modelo:** Precisión, F1-Score, Recall y Pérdida registrar estas métricas en CloudWatch para monitoreo constante y configurar umbrales de alerta si los valores bajan. Grafana puede visualizar cambios en las proporciones de clases. Un cambio significativo puede ser una señal de que los datos están cambiando y afectar la precisión del modelo.

3. **Sistema de Alertas para Problemas de Calidad del Modelo**

   - **Errores en la Ejecución del Pipeline:** Usar Amazon CloudWatch Alarms para enviar alertas en caso de error en la ejecución de jobs críticos o en GitHub Actions. Notificaciones en Slack, Email o SMS mediante Amazon SNS para informar al equipo de fallos importantes en tiempo real.
   - **Desempeño del Modelo por Debajo del Umbral:** Configurar alarmas para detectar cuando el rendimiento del modelo disminuya por debajo de un umbral, lo que podría indicar una deriva en los datos de entrada o en la efectividad del modelo.

4. **Automatización de Acciones Correctivas**

   - **Reentrenamiento Automático:** Configurar AWS Lambda o GitHub Actions para que actúen como disparadores de reentrenamiento automático si las métricas del modelo o las características de los datos cambian significativamente. Definir reglas en Step Functions para ejecutar automáticamente el pipeline de reentrenamiento.
   - **Ajuste de Parámetros Automático:** Usar SageMaker Hyperparameter Tuning para ajustar automáticamente los parámetros del modelo en caso de caída en métricas de rendimiento. Integrar un flujo en Step Functions o Lambda para gestionar los ajustes y seleccionar el mejor modelo basado en los resultados de las pruebas.
   - **Reinicio del Pipeline en Caso de Fallos Transitorios:** Configurar AWS Step Functions para reintentar automáticamente pasos fallidos del pipeline que sean propensos a errores transitorios, como conexiones interrumpidas o falta de recursos temporales.
   - **Notificación de Resultado Correctivo:** Informar automáticamente al equipo sobre el ajuste realizado en los parámetros, el reentrenamiento completado o cualquier otro cambio correctivo importante mediante SNS.

Propuesta de Seguridad y Guardrails

- **Liste los componentes de seguridad que utilizaría para asegurar que el pipeline de NLP y el modelo sean robustos.**

1. **Autenticación y Autorización**

   - **AWS IAM:** Definir roles y permisos específicos para limitar el acceso a servicios como S3 y SageMaker, asegurando que solo los usuarios y servicios autorizados tengan acceso.
   - **Control de Acceso en GitHub:** Usa autenticación de dos factores y permisos mínimos para proteger el código del pipeline.

2. **Encriptación de Datos**

   - **Encriptación en Tránsito y en Reposo:** Usar HTTPS para transmisión de datos y **AWS KMS** para encriptar datos en reposo en S3, RDS, o DynamoDB.

3. **Seguridad en el Despliegue**

   - **VPC y Subredes Privadas:** Ejecutar el pipeline en una VPC privada para evitar accesos externos.
   - **AWS Security Groups y ACLs:** Configurar para permitir solo tráfico autorizado.
   - **Auditoría con AWS CloudTrail:** Registrar accesos y modificaciones en AWS.

4. **Protección contra Manipulación y Verificación de Integridad**

   - **Control de Versiones:** Usar GitHub para monitorear y revertir cambios en el código.
   - **Almacenamiento Inmutable:** Almacenar modelos aprobados en S3 con Object Lock para evitar modificaciones no autorizadas.

5. **Seguridad de Datos del Modelo e Inferencias**

   - **Anonimización de Datos:** Aplicar técnicas de anonimización para proteger datos de usuario.
   - **Políticas de Retención de Datos:** Definir políticas para eliminar datos obsoletos y reducir exposición.

6. **Detección y Respuesta a Incidentes**

   - **Amazon GuardDuty y AWS Config:** Detección de amenazas y monitoreo de configuraciones.
   - **Alertas con CloudWatch y SNS:** Notificaciones automáticas en caso de accesos no autorizados o fallos.