Prueba para Machine Learning Engineer

Autor: Juan Sebastian Arbelaez

1. Propuesta de Arquitectura en la nube para un modelo NLP

A la hora de decantarse por un tipo de arquitectura u otro, se encuentran diferentes factores a tener en cuenta. Para este caso en específico, se habla directamente de una arquitectura para inferencias en batch, lo que ya acota un poco el problema. Sin embargo, siguen existiendo temas de capacidad de cómputo, qué tan ambiciosos queremos ser con lo que queremos predecir y por supuesto también el factor de gastos o recursos económicos provistos para el proyecto.

Por esa razón es que parto por proponer dos tipos diferentes de arquitectura, a pesar de que grosso modo su objetivo es atacar el mismo tipo de problema, durante el proceso de cada una existen ciertas características que las hacen diferentes la una de la otra y pasaré a mencionar a medida que avancemos con esta explicación:

1.1. Arquitectura basada en serverless con AWS Lambda.

Para esta primera opción, se propone un flujo fundamentado en tecnologías serverless (con el objetivo de aprovechar las ventajas del pay as you go). El flujo principal de esta opción es el siguiente: AWS Lambda procesa los textos en batches y ejecuta las inferencias usando un modelo de NLP que previamente hemos almacenado en AWS S3. Ahora, la pregunta es cómo orquestamos el proceso? A través de AWS Step Function. En caso de necesitar hacer preprocesamientos complejos, también encontramos a AWS Glue en la ecuación.

Entremos a analizar entonces, las ventajas y desventajas que pueden surgir de esta arquitectura:

- Ventajas:
    - Escalable (Lambda se puede ejecutar en paralelo sin gestionar servidores)
    - Bajo costo (Pagamos por invocar las lambdas y por el uso de S3, sin infraestructura persistente)
    - Fácil de mantener (AWS gestiona la infraestructura y actualizaciones)

- Desventajas:

    - Límite de tiempo (Lambda tiene un máximo de ejecución de 15 min.)
    - Memoria (No apto para modelos grandes que exigen RAM y CPU)
    - Si los textos son muy grandes, resulta mejor otra opción con mayor capacidad de cómputo.

Pasemos a evaluar entonces ahora la segunda pregunta que hace alusión a la arquitectura propuesta en términos de escalabilidad y confiabilidad frente a grandes volúmenes de datos.

La arquitectura severless basada en AWS Lambda y Step Functions escala de manera automática sin necesitar de la gestión de servidores. Esto significa que Lambda puede ejecutarse en múltiples instancias concurrentes (hasta 1000 simultaneas). Aquí el papel de Step Functions es dividir el procesamiento en pasos manejables, asegurando que cada etapa se complete antes de pasar a la siguiente (se encarga de orquestar el proceso). En términos de tolerancia a fallos, AWS Lambda maneja estos fallos reiniciando automáticamente las ejecuciones fallidas, y para hacer un double check, Step Functions permite definir estrategias de reintento.

Sin embargo, esta arquitectura sí puede presentar limitaciones a la hora de hablar de grandes volúmenes de datos, en términos de memoria y tiempo, ya que Lambda está limitado a 15 minutos por ejecución y máximo el uso de 10GB de RAM, lo que pone esta arquitectura en desventaja frente a grandes volúmenes de datos.

Esta arquitectura probablemente fallaría frente a modelos pesados o muy grandes de NLP, aunque se podría optar por algunas estrategias para mitigar en caso de que el modelo NLP no fuera demasiado pesado:

- División del procesamiento en lotes pequeños dentro de Step Functions
- Optimización (cuantización, compresión)
- Preprocesamiento eficiente para reducir el tamaño del texto antes de inferir

Analicemos por componente su papel en la arquitectura:

Amazon S3: Almacena los datos de entrada (archivos JSON, CSV o Parquet con textos). Guarda también los resultados del modelo.

AWS Lambda: Ejecuta la inferencia del modelo en lotes pequeños. Se invoca cada vez que hay nuevos archivos en S3 o en intervalos de tiempo definidos.

AWS Step Functions: Coordina el flujo de trabajo, permitiendo manejar múltiples lotes en paralelo. Asegura que cada tarea se complete antes de pasar a la siguiente.

AWS Glue: Si los textos requieren preprocesamiento complejo (limpieza, tokenización avanzada, etc.), AWS Glue puede ejecutar scripts de transformación en Spark o PySpark.

Entonces, para finalizar el análisis de esta opción:

Esta opción es ideal para procesamientos de texto de batches pequeños (menos de 50.000 textos por hora), que incluya modelos NLP ligeros que puedan ejecutarse en menos de 15 minutos y donde el bajo costo sea primordial para nuestra aplicación. En caso de cumplir con estos requisitos, me decantaría por un tipo de opción como esta.

1.2. Arquitectura basada en AWS SageMaker con Batch Transform

Como segunda opción, en caso de que definitivamente necesitemos hacer uso de modelos de NLP grandes y exigentes en términos computacionales, o que nuestra cantidad de textos por hora sea alta y no podamos recurrir a la opción con Lambda, se hace esta propuesta.

Para esta arquitectura, utilizamos AWS SageMaker Batch Transform, que es un servicio diseñado específicamente para inferencias por batches con modelos pre-entrenados. Esta opción es una opción mucho más robusta, pero a su vez trae consigo otras cuestiones a tener en cuenta para llevar a cabo su implementación. Pero si definitivamente estamos hablando de modelos NLP de gran tamaño y gran prosamiento, esta sería nuestra arquitectura adecuada.

Haciendo el mismo tipo de análisis de la propuesta anterior, entremos a ver sus ventajas y desventajas.

- Ventajas:

    - Óptimo para NLP en batch ya que Batch Transform es ideal para aplicar a modelos con grandes volúmenes de texto.
    - Su escalabilidad es automática ya que SageMaker gestiona los recursos de cómputo según el tamaño del dataset.
    - No encontramos límites en el tiempo de ejecución a diferencia de Lambda.

- Desventajas:

    - Los costos son más altos, a pesar de que SageMaker cobre por uso, puede llegar a ser más caro que una solución serverless.
    - Es necesaria la administración, definiendo estancias y gestionando los modelols.

Evaluemos entonces nuevamente en términos de escalabilidad y confiabilidad para grandes volúmenes de datos. SageMaker Batch transform está diseñado específicamente para procesamiento en batch y por ende puede manejar grandes volúmenes de datos con modelos pesados.

En términos de escalabilidad, SageMaker ajusta automáticamente los recursos de cómputo, permitiendo procesar millones de registros sin intervención manual. Adicionalmente, se puede utilizar múltiples instancias para dividir el trabajo y acelerar las inferencias.

Como está pensado para soportar modelos grandes, también permite la ejecución de modelos que requieran GPUs o grandes cantidades de memoria, y como hablamos en este caso de almacenar nuevamente datos en S3, no hay restricción en el tamaño de éstos en la entrada ni en la salida (resultados).

Para garantizar la confiabilidad, SageMaker reintenta trabajos fallidos y, como ya mencioné, puede distribuir la inferencia en varias instancias evitando puntos únicos de fallo.

Analicemos por componentes la arquitectura:

Amazon S3: Almacena los datos de entrada y salida. También almacena los modelos entrenados en formato tar.gz.

AWS Glue: Preprocesa datos en batch antes de la inferencia (tokenización, eliminación de ruido, normalización).

Amazon SageMaker Batch Transform: Ejecuta inferencias en lotes utilizando múltiples instancias en paralelo.

AWS Step Functions: Orquesta el flujo de ejecución de preprocesamiento, inferencia y post-procesamiento (en caso de tener dependencias de unos pasos con otros).

Amazon Athena / Redshift: Si se requiere análisis posterior de los resultados, estos se pueden consultar a través de Athena o Redshift dependiendo de nuestra arquitectura de datos (Data Lake o DataWarehouse).

AWS EventBridge: Dispara automáticamente el proceso en intervalos de tiempo programados.

Para finalizar el análisis de nuestra segunda opción:

Se recomienda esta arquitectura para cuando se procesan millones de textos en lotes grandes. Además, si el modelo de NLP es grande y complejo (como un BERT, o GPT). Esta opción cuenta con soporte para modelos que requieran de GPU o en donde se busque mejorar la velocidad a través de esta.

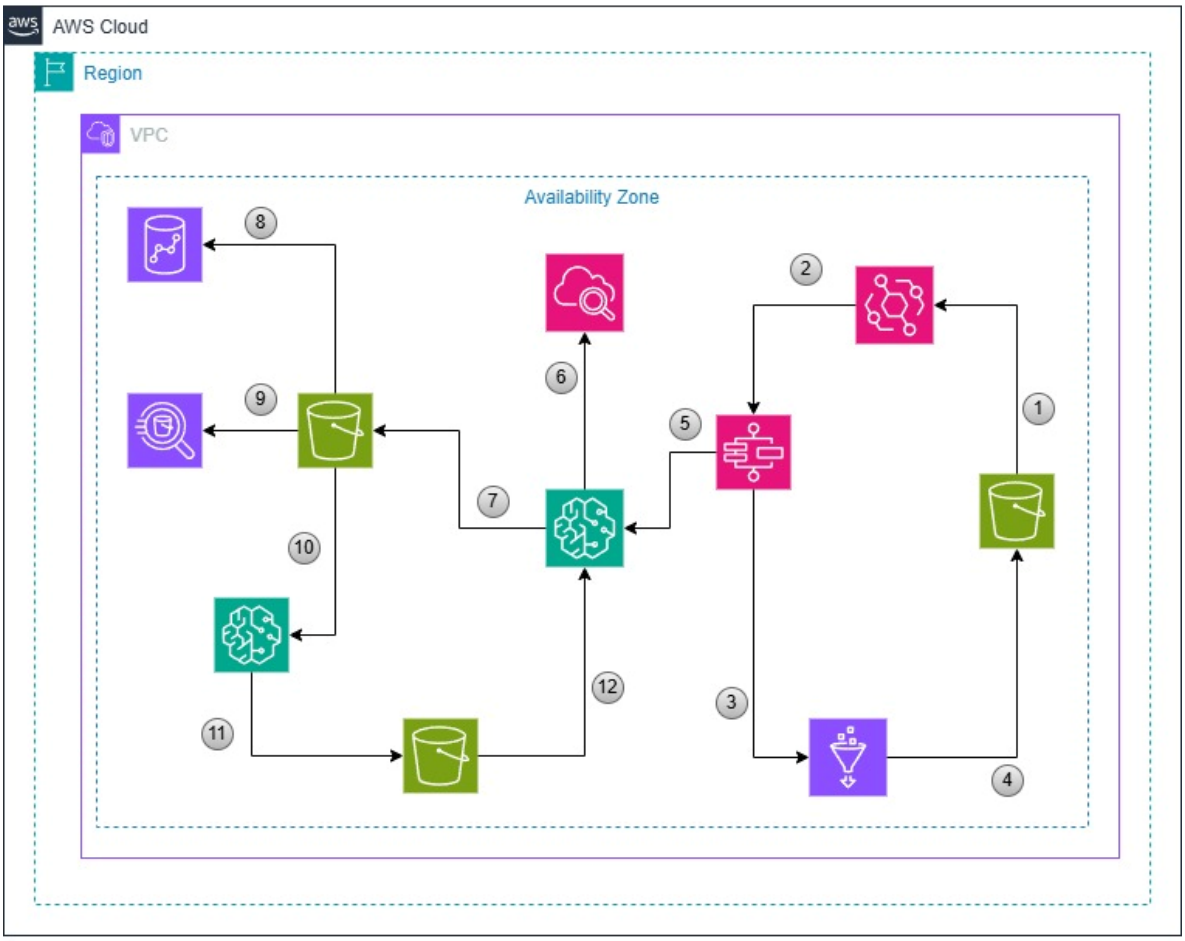
Finalmente:

Si los volúmenes de datos son pequeños o medianos y buscas bajo costo, usa Lambda con Step Functions.

Si procesas millones de textos con modelos NLP grandes, usa SageMaker Batch Transform, ya que está diseñado específicamente para inferencia en batch.

2. **Diagrama de Arquitectura:**

Diagrama de arquitectura



1. Nuevo dataset disponible
2. Disparar ejecución del pipeline
3. Ejecutar preprocesamiento de datos
4. Guardar datos preprocesados
5. Ejecución del modelo NLP en batch
6. Logs y monitoreo
7. Resultados de inferencia
8. Consulta y analisis en Redshift
9. Consulta y analisis en Athena
10. Datos usados para entrenamiento continuo
11. Actualización y registro de nuevo modelo
12. Carga nuevo modelo entrenado

3. **Componente de Orquestación:**

A ctinuacion, procederé a explicar la orquestación del sistema que está fundamentada en AWS Step Functions, el cual es clave para coordinar y automatizar la ejecución del pipeline de NLP.

Como primer labor, el orquestador tiene como objetivo disparar y coordinar la ejecución del pipeline. Para este caso en particular, podemos tener 2 tipos diferentes de activación a través de EventBridge:

1) Cron-job (con X periodicidad)
2) En respuesta a un evento (llegada de un archivo a S3)

Una vez se activa el orquestar, comienza la "sinfonía" en un flujo secuencial, donde el orquestador se encarga de organizar cada paso en un orden lógico, asegurando que no se ejecuten tareas antes de que las anteriores se completen de manera exitosa. Sin ir muy lejos, veamos un ejemplo: si los datos no han sido preprocesados correctamente en Glue, entonces el orquestador detendrá la ejecución del pipeline en vez de enviar datos incorrectos a SageMaker.

Es decir, Step Functions sirve para gestionar las dependencias entre los diferentes componentes del sistema, manejando las relaciones entre los diferentes servicios usados en el pipeline. Por mencionar algunos pasos:

- Espera a que Glue termine de preprocesar antes de ejecutar SageMaker
- Asegura la carga de resultados a Redshift o a Athena solo si la inferencia fue exitosa
- Puede dividir tareas en paralelo si hay, por ejemplo, múltiples modelos que daban ejecutarse al mismo tiempo.

Otra de las tareas que lleva a cabo el orquestador es la de manejar fallos y reintentos de manera automática. Es muy importante este paso ya que uno de los beneficios más grandes que nos proporciona Step Functions es su capacidad de manejo de errores:

- Reintentos automáticos: Si Glue o SageMaker fallan, el orquestador puede reintentar la tarea según reglas definidas (por ejemplo intentar 3 veces en intervalos de 1 minuto)
- Tolerancia a fallos: Si un paso falla repetidamente, Step Functions puede hacerse cargo de enviar una alerta y detener la ejecución.

Eso es básicamente lo que hace el orquestador en esta arquitectura propuesta.

La estructura de carpetas que propondría para un proyecto así estaría basada en clean architecture y con una orientación modular:

```
nlp_batch_pipeline/
├── config/                    # Configuración y variables globales
├── data/                      # Datos utilizados en el pipeline
│   ├── raw/                   # Datos sin procesar
│   ├── processed/             # Datos preprocesados listos para inferencia
│   ├── results/               # Resultados de inferencia
├── src/                       # Código fuente del proyecto
│   ├── preprocessing/         # Módulos de preprocesamiento
│   ├── inference/             # Implementación del modelo y ejecución en batch
│   ├── postprocessing/        # Procesamiento de resultados de inferencia
│   ├── training/              # Scripts de reentrenamiento del modelo
│   ├── aws_integration/       # Código para interactuar con AWS
├── scripts/                   # Scripts ejecutables para lanzar procesos
├── tests/                     # Pruebas unitarias y de integración
├── logs/                      # Archivos de logs de ejecución
├── notebooks/                 # Notebooks para pruebas y análisis
├── deployment/                # Infraestructura como código (IaC)
├── requirements.txt           # Dependencias del proyecto
├── Dockerfile                 # Configuración de imagen Docker
├── README.md                  # Documentación del proyecto
├── .gitignore                 # Archivos a ignorar en Git
├── Makefile                   # Atajos para ejecutar tareas comunes
```

OJO: En la sección anterior la carpeta “data”, realmente la destinaría a guardar las rutas por ejemplo hacia S3 de los archivos requeridos para ejecutar el programa, ya que no es una buena práctica alojar archivos en repositorios.

Con respecto al versionamiento del preprocesamiento y del modelo, procuraría utilizar una herramienta como MLFlow para tener un registro ordenado. En caso de no ser posible, propondría una solución más artesanal basada en directorios de S3 de una manera muy organizada, estandarizando los nombres de los archivos con versiones y fechas para un mejor control.

4. **Propuesta de Monitoreo del modelo en batch**

Cuando hablamos de monitoreo nos referimos a la parte crítica que se encarga de detectar problemas de calidad en nuestro modelo, el rendimiento y la estabilidad de la infraestructura. En un esquema sólido de monitoreo, debemos contar con métricas claves, sistemas de alerta y automatización de acciones correctivas.

Para monitoreo la arquitectura propuesta a partir de procesamiento batch en AWS, podemos echar mano de diferentes herramientas a lo largo de nuestro flujo.

En primer lugar, ya habíamos hablado de integrar CloudWatch para capturar logs, ahí mismo podemos capturar métricas y alertas. Adicionalmente, contamos con Step Functions que posee herramientas como Step Functions Metricas para supervisar la ejecución del pipeline y detectar los fallos (además de las bondades ya mencionadas en reintentos fallidos que posee Step Functions). Desde el punto de vista de SageMaker, éste cuenta con una herramienta llamada Model Monitor para evaluar el rendimiento del modelo y detectar la degradación del mismo por temas de drift. Y finalmente, si queremos enviar alertas, podemos utilizar el sistema de notificaciones simples de amazon conocido como Amazon SNS (para casos de errores).

Métricas relevantes a monitorear:

A la hora de hablar de métricas relevantes a monitorear podemos dividirlas en tres segmentos diferentes.

1) Métricas de Infraestructura

En este caso buscamos garantizar que el pipeline se ejecute sin interrupciones.

- Tiempo de ejecución del pipeline (Cuanto tarda en completarse)
- Errores en Step Functions (si falló alguna etapa del pipeline)
- Consumo de CPU/RAM en SageMaker (problemas de recursos)
- Costo del pipeline

Para estos casos una acción correctiva sería CloudWatch Trigger para volverlo a ejecutar.

2) Métricas de calidad del Modelo

Aquí lo que buscamos observar es si el modelo sigue funcionando correctamente.

- Distribución de predicciones (% de reseñas positivas vs negativas ejemplo)
- Score de confianza del modelo (probabilidad promedio de predicciones)
- Data drift (si los datos de entrada han cambiado)
- Desempeño (Accuracy, Precision, Recall)

Para este caso particular la acción correctiva sería que en caso de detectar data drift o bajo desempeño, se puede disparar un nuevo entrenamiento automáticamente.

3) Métricas de integridad de los datos

Por último nos interesa detectar si los datos de entrada son inconsistentes o tienen problemas.

- Tamaño del dataset (Detectar si faltan datos o los lotes son incorrectos)
- Filas con datos faltantes
- Formatos incorrectos

Aquí tocaría avisar al equipo de datos y bloquear la ejecución del piepeline hasta corregirlo.

Sistema de alertas:

Para el sistema de alertas se propone utilizar Amazon CloudWatch y Amazon SNS de la siguiente manera:

- Error en Step Functions (Enviar notificación email/slack)
- Baja precisión del modelo (Enviar notificación email/slack)
- Exceso de CPU en SageMaker (Enviar notificación email/slack)
- Data drift detectado (Enviar notificación email/slack)

Como mencioné anteriormente, para no depender de la intervención manual, podemos automatizar la respuesta a ciertas métricas: Reentrenamiento del modelo, reinicio del pipeline si falla y más.

5. **Propuesta de Seguridad y Guardrails**

El componente de seguridad es fundamental a la hoar de proteger datos, el modelo y los accesos a diferentes recursos que tenemos en AWS.

Es necesario tener presente la seguridad en unas etapas fundamentales del flujo:

- La seguridad en los datos: Con el objetivo de proteger los datos de entrada, salida y preprocesados, en este caso, para S3. En un principio tomando acciones como controles de acceso a través de IAM, generando permisos en forma granular, partiendo del principio de los mínimos permisos necesarios a través de políticas, roles y grupos para acceso a los servicios y datos. Adicionalmente, el bloqueo de acceso publico a S3 para evitar exposición de datos. Adicionalmente, nos podemos apoyar en CloudTrail para capturar y registrar cada acceso a los datos de S3. Y finalmente, opciones más avanzadas como temas de encriptación que deben ir de la mano con los equipos de ciberseguridad como S3 Server-Side Encryption SSE-S3 o AWS Key Management Service KMS.

- Seguridad en la orquestación: El mismo tema de principio de los mínimos privilegios, y que invoque exclusivamente los servicios específicos del pipeline. Utilizar también IAM Roles temporales en lugar de credenciales fijas, que tengan una variación en el tiempo. Por último, configurar el Step Functions con ExecutionTimeout y MaxConcurrentExecutions para evitar sobrecarga de recursos. Podemos ver como Guardrail que si el pipeline falla 3 veces seguidas, se bloquea la ejecución y se notifica al equipo de seguridad.

- Seguridad en la inferencia y en el modelo: La idea aquí sería evitar fugas de datos y asegurar que el modelo se ejecute en un entorno seguro. Volvemos al tema de IAM Policies o Roles para restringir quien puede ejecutar SageMaker. También se puede optar por limitar el tamaño y número de consultas al modelo. Adicionalmente se puede encriptar y aislar en una VPC privada, eliminar el acceso a internet para evitar extracciones y se pueden cifrar los datos del modelo con KMS como ya mencioné anteriormente. Guardral sería que si se detecta trafico sospechoso se desactiva la inferencia automáticamente.