UNIVERSIDAD DE ANTIOQUIA

Diseño e implementación de un sistema reactivo para análisis de sentimientos con aprendizaje automático

Autor(es)

Carlos Augusto Hincapié Romero

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Ingeniería de Sistemas

Medellín, Colombia

2021

CC BY NC SA

Diseño e implementación de un sistema reactivo para análisis de sentimientos con aprendizaje automático

Carlos Augusto Hincapié Romero

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:
Ingeniero de sistemas

Asesores (a):
ANTONIO JESÚS TAMAYO HERRERA - M. Sc. Ing.

Línea de Investigación:
Ingeniería de Software
Grupo de Investigación:
In2Lab

Universidad de Antioquia
Facultad de Ingeniería, Departamento de ingeniería de sistemas
Medellín, Colombia
2021

Diseño e implementación de un sistema reactivo para análisis de sentimientos con aprendizaje automático

Resumen

Los sistemas de aprendizaje automático requieren del uso de arquitecturas de software que permitan implementar el flujo de ingesta, entrenamiento y publicación de modelos de una forma resiliente, elástica y responsiva. Una arquitectura de microservicios reactiva permite dar solución a los retos de usabilidad y técnicos que los sistemas de aprendizaje automático imponen. En este trabajo se describe la solución a un problema de procesamiento de lenguaje natural, análisis de sentimientos, utilizando los principios de arquitecturas de microservicios reactivas, y se utilizan los servicios de amazon web services (aws) para implementar dicha solución.

Palabras clave:

Aprendizaje automático, arquitectura reactiva, Python, microservicios, procesamiento de lenguaje natural, AWS.

Introducción

Un sistema de aprendizaje automático es un conjunto de componentes de software que tienen la capacidad de aprender de los datos y hacer predicciones basados en nuevas entradas, en 2015 Bell[1] [1] definió el aprendizaje automático así: "El aprendizaje automático es una rama de la inteligencia artificial. Utilizando la informática, diseñamos sistemas que pueden aprender de los datos en forma de datos entrenados. Los sistemas pueden aprender y mejorar con la experiencia y con el tiempo, refinar un modelo que se puede usar para predecir resultados sobre nuevas preguntas basadas en el aprendizaje previo."
Un flujo de trabajo secuencial para un sistema de aprendizaje automático puede verse en la figura 1.

[1] El texto original de esta referencia está escrito en inglés

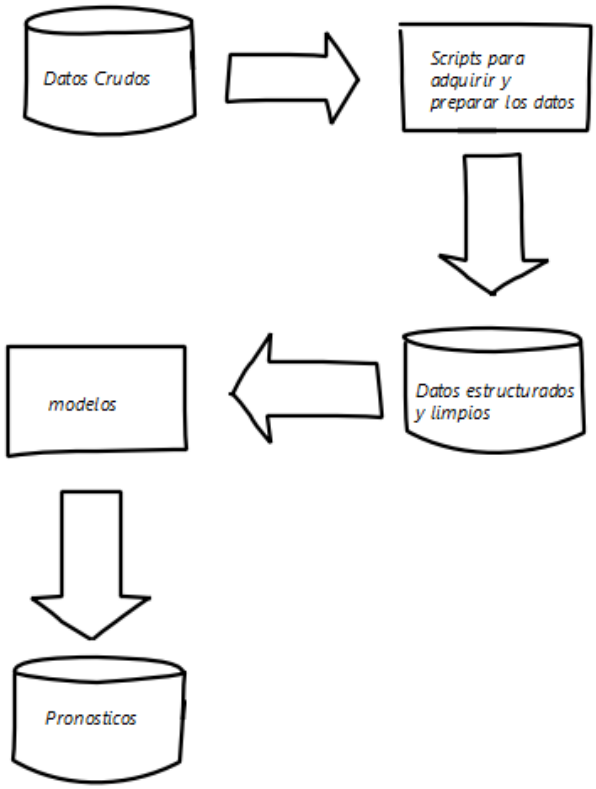


Figura 1 - Flujo de trabajo básico para para un sistema de aprendizaje automático

Desde diferentes sistemas de información, se vuelcan los datos necesarios para el entrenamiento del modelo, hacia un repositorio de información no estructurado que en este documento se llama "Datos Crudos"; después se toma esta información utilizando programas tipo script, cuya responsabilidad es adquirir y preparar los datos. Del proceso de extracción, transformación y limpieza de datos se obtiene una información estructurada y lista para ser utilizada en el entrenamiento. Posteriormente se selecciona el modelo más óptimo para el problema a resolver (modelos supervisados, modelos no-supervisados, modelos paramétricos, modelos no-paramétricos, problemas de regresión, problemas de clasificación, etc.). Esto es, el modelo que tenga el mejor resultado para predecir el valor de una o varias variables de salida sobre un conjunto de datos de entrenamiento. Finalmente, el modelo seleccionado se usa sobre un conjunto de datos nuevo, sobre los cuales se desea efectuar un pronóstico.

Sin embargo, el flujo de trabajo de la figura 1 sólo funciona correctamente en un sistema de aprendizaje automático sencillo, en un sistema industrial de aprendizaje automático es necesario un reentrenamiento de los modelos de manera rutinaria. Smith[1] [2] afirma, ".. el científico de datos se dio cuenta de que las predicciones no cambiaban mucho, por lo que le preguntó al ingeniero sobre la frecuencia de reentrenamiento de los scripts secuenciales de modelado. El ingeniero no tenía idea de qué estaba hablando el científico de datos. Eventualmente se dieron cuenta de que el científico de datos tenía la intención de que sus

scripts se ejecutan a diario sobre los datos más recientes del sistema. Todos los días debería haber un nuevo modelo en el sistema para reemplazar el anterior".
En el flujo de trabajo de la figura 2, se observa un reentrenamiento del modelo con nuevos datos de forma periódica, de tal forma que cada tanto se crea un nuevo modelo y se reemplaza el existente.

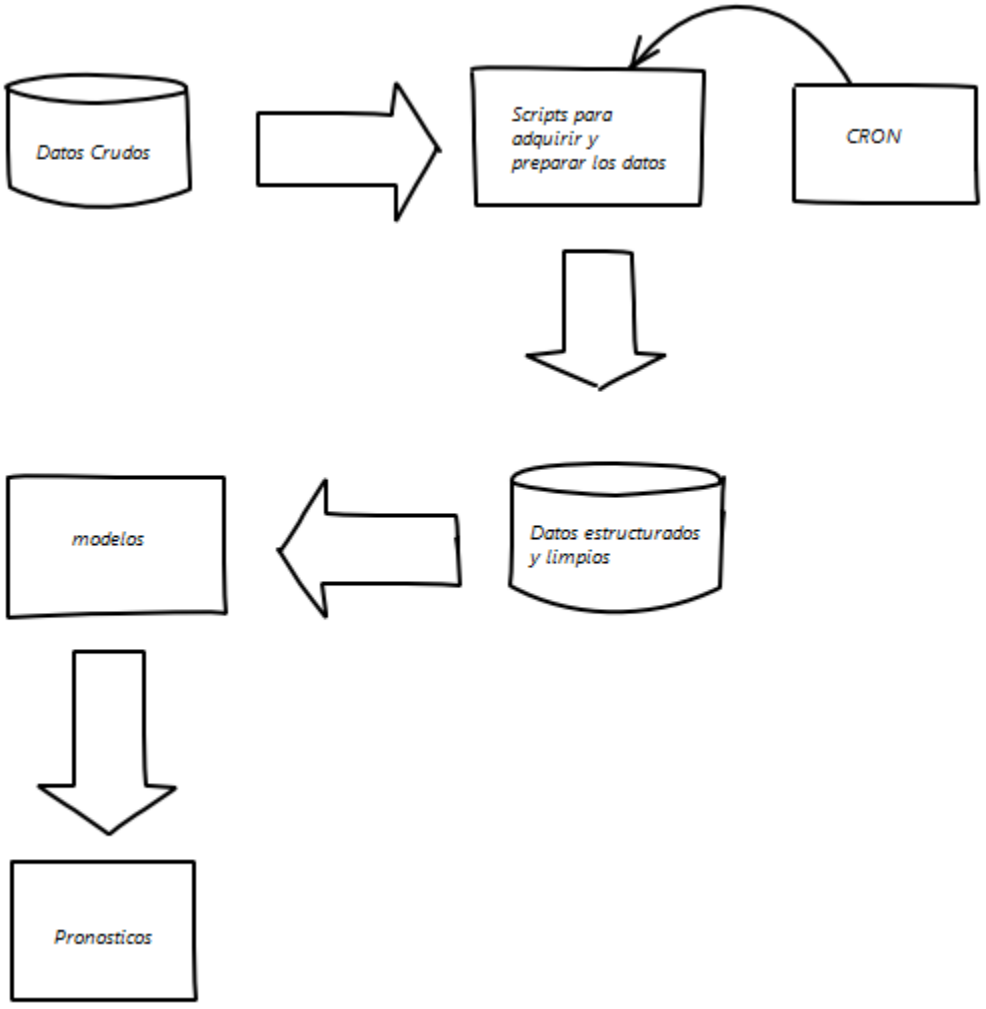


Figura 2- Flujo de trabajo con re-entrenamiento para sistema de aprendizaje automático

Estos sistemas están sometidos a una alta carga de información susceptibles de fallas en los procesos de extracción, limpieza, transformación y carga de la información histórica necesarios para el entrenamiento del modelo, y además el tamaño de los conjuntos de datos usados para la optimización y prueba de los modelos se incrementa en cada ciclo de re-entrenamiento. En 2018 Smith[1] [2] afirmó, "Enviar los datos de la aplicación al servidor requería una transacción. Cuando el científico de datos y el ingeniero agregaron más datos a la cantidad total de datos que se recopilaron para el modelado, esta transacción tomó demasiado tiempo para mantener una capacidad de respuesta razonable dentro de la aplicación. La función de predicción dentro del servidor que admitía la aplicación no manejaba las nuevas funciones correctamente. El servidor lanzaba una excepción cada vez

que la función de predicción detectaba alguna de las funciones nuevas que se habían agregado en otra parte de la aplicación".

Cuando solamente se le entrega la responsabilidad del entrenamiento del modelo a un proceso, este puede fallar y toda la información perderse. Estas fallas son difíciles de detectar sin construir un sistema de monitoreo y alertas en la infraestructura, pero inclusive si se detecta una falla en el sistema, lo único que se podría hacer es iniciar la tarea de nuevo y esperar que tenga éxito en el nuevo intento. Sin embargo, las modificaciones en los procesos de extracción, transformación y carga (ETL), son difíciles de realizar sin afectar el desempeño de todo el modelo. Los modelos de aprendizaje automático a menudo requieren modificar las variables de entrada y para esto es necesario que las aplicaciones capturen más información de los usuarios; si el número de usuarios crece y las aplicaciones no están preparadas para esto, se pueden presentar problemas de latencia en el envío de la información y degradación general de todo el sistema, dejando al sistema y a la infraestructura que lo soporta, inhabilitada para un buen funcionamiento.

En resumen, los sistemas de aprendizaje automático reactivo deben cumplir con las condiciones del "*Reactive Manifest*", en 2018, Smith [2], propuso que para un funcionamiento confiable un sistema debe poseer los siguientes atributos:

- El sistema debe mantenerse sensible (Responsive), aunque exista algún problema en los componentes del sistema encargados de la predicción.
- Los componentes de predicción del sistema deben estar desacoplados del resto de unidades del sistema.
- El sistema de predicciones debe comportarse de forma previsible sin importar la alta carga del sistema mismo.
- El sistema debe permitir una fácil modificación de los componentes sin que se ocasionen daños colaterales a otras partes.
- El sistema debe ser capaz de medir su desempeño de modelado.
- El sistema debe soportar la evolución y el cambio.
- El sistema debe soportar la experimentación en línea.
- Las personas podrán supervisar fácilmente el sistema predictivo y corregir cualquier mal comportamiento.

La solución a estos desafíos se elabora pensando en los sistemas de aprendizaje automático como una aplicación y no solamente como una técnica.

En este trabajo se diseñó y se implementó un sistema de aprendizaje automático para análisis de sentimientos, que consiste en clasificar un texto como positivo o negativo. El sistema se diseñó siguiendo los principios de las **aplicaciones reactivas**; este tipo de sistemas tiene como característica fundamental de diseño, la reacción adecuada a los cambios de ambiente. Para ello, estas aplicaciones cumplen con cuatro características básicas: son sensibles (Responsive), resilientes (Resilent), elásticas (Elastic) y la comunicación entre los módulos que componen está basada en mensajes (Message - Driven), todo esto acompañado con estrategias de replicación, contención y supervisión de componentes.

Para implementar esta arquitectura se utilizó Python como lenguaje de programación, una arquitectura basada en microservicios reactivos que sigue en la propuesta de Bonér [5], y AWS [21] cómo plataforma en la nube para el desarrollo y el despliegue de los componentes.

Objetivos

Objetivo General

Desarrollar un sistema con una arquitectura de microservicios, siguiendo los paradigmas de los sistemas reactivos que permita hacer análisis de sentimientos de reseñas de productos o servicios en español.

Objetivos Específicos

- Implementar un sistema de aprendizaje automático y procesamiento de lenguaje natural (PLN) para la clasificación de opiniones sobre reseñas de productos o servicios en español.
- Diseñar una arquitectura de software reactiva utilizando patrones de microservicios reactivos para solucionar los requisitos no-funcionales asociados a sensibilidad (Responsive), resiliencia (Resilient), elasticidad (Elastic), comunicación segura entre los componentes basada en mensajes (Message - Driven) utilizando las estrategias de replicación, contención y supervisión de componentes.
- Utilización del lenguaje de programación Python para la implementación.
- Simular diferentes ambientes para probar el funcionamiento del sistema.
- Desplegar la aplicación en la nube.

Marco Teórico

Mejora continua del modelo

Un sistema de aprendizaje automático mejorará continuamente el modelo re-alimentándose de la información capturada por el sistema mismo, de esta forma se adaptará a cambios en los datos. En 2017, Brink[1] [3] afirmó: *"*La mayoría de los modelos ML tradicionales son estáticos o rara vez se reconstruyen. Pero en muchos casos, los datos y las predicciones fluyen de regreso al sistema y se desea que el modelo mejore con el tiempo y se adapte a los cambios en los datos. Varios algoritmos de aprendizaje automático admite este tipo de aprendizaje en línea".

Un sistema de aprendizaje automático que se autoalimenta a sí mismo pasará por los siguientes ciclos: recolección de datos, generación de características, construcción del modelo, optimización del modelo, publicación del modelo y predicciones de la información, cuando termine la última etapa volverán de nuevo a repetirse cada una de las etapas en el mismo orden de forma indefinida [3].

Los sistemas de aprendizaje automático requieren el consumo de grandes conjuntos de datos (*datasets*), la recepción de información a alta velocidad, además los módulos que reciben, modelan y procesan los datos requieren una alta confiabilidad, para reducir al mínimo los periodos de indisponibilidad [2].

En la actualidad la información que utilizan los sistemas de aprendizaje automático puede provenir de sensores, páginas web, blogs y todo tipo de dispositivos móviles.

Para soportar requisitos no-funcionales como disponibilidad, desempeño, confiabilidad, escalabilidad y modificabilidad los sistemas de aprendizaje automático requieren diferentes estrategias que los diferencian de las aplicaciones transaccionales tradicionales en donde

estos requisitos no funcionales son menos exigentes, y sobre todo más predecibles en el tiempo.

Optimización del modelo

Brink [3], propuso que se puede alcanzar un mejor desempeño del modelo a través de tres métodos:

- Afinando los parámetros del modelo: Los valores de cada parámetro dependen del tipo y estructura de los datos, el valor de cada parámetro o la combinación de cualquiera de ellos tiene impacto en el desempeño del modelo, se aplican métodos para seleccionar la mejor combinación de parámetros posible para el conjunto de datos seleccionado.
- Seleccionando el conjunto de características: Muchos problemas de aprendizaje automático incluyen un gran número de características, algunas de las cuales no son relevantes para el modelo, estas características irrelevantes producen ruido en los resultados pronosticados, una labor importante es seleccionar las características que mejor contribuyen al desempeño del modelo.
- Pre-procesando los datos: La mayor parte de los conjuntos de datos origen no se encuentran en perfecto estado, muchos datos pueden incluir nombres que son deletreados de forma diferente, aunque se refieren al mismo concepto o también pueden tener valores vacíos o valores incorrectos; para todos estos casos es necesario realizar limpieza y procesamiento, este proceso es conocida como "*data munging*" o "*data wrangling*".

La arquitectura de software para aplicaciones de aprendizaje automático necesita responder a dos retos:

1. Elevados requisitos no-funcionales cómo disponibilidad, desempeño, confiabilidad, escalabilidad y modificabilidad.
2. Un hardware de servidores y redes mucho más complejo que las aplicaciones transaccionales tradicionales de hasta hace unos pocos años.

Ayer	Hoy
Máquinas únicas	Clúster de Máquinas
Procesadores únicos	Múltiples procesadores
Costo alto de la memoria ram	Costo relativamente bajo de memoria ram
Memoria física costosa	Memoria física económica
Redes de comunicación lentas	Redes de comunicación rápidas
Pocos usuarios concurrentes	Gran cantidad de usuarios concurrentes
Conjuntos de datos pequeños	Grandes conjuntos de datos
Latencia aceptable de segundos	Latencia aceptable en milisegundos

Tabla 1 - Evolución de la infraestructura física

Para solucionar los elevados requisitos no-funcionales que estos sistemas demandan y las ventajas que actualmente ofrecen los centros de datos con clusters de servidores de

múltiples núcleos, se concibió en años recientes el modelo de desarrollo basado en arquitecturas reactivas de microservicios.
El modelo de microservicios reactivos define una forma de diseñar e implementar aplicaciones que distribuyen su trabajo entre todos los recursos del sistema, desde hilos, núcleos, clusters de servidores y centros de datos.
El modelo de microservicios reactivos fue creado para proporcionar una forma efectiva de construir aplicaciones con alto nivel de concurrencia e incrementar el uso eficiente de recursos. Además, el modelo de microservicios reactivos también define maneras bien definidas para manejar las fallas en forma elegante, por lo tanto, los sistemas basados en microservicios tienen niveles de resiliencia elevados que permiten aislar problemas y evitar fallas en cascada. En 2017, Bonér[1] [5] afirmó, "..Entonces, es hora de despertar, de retirar el monolito y de descomponer el sistema en servicios discretos y manejables que se pueden escalar individualmente, que pueden fallar, implementar y actualizar de forma aislada.
Ellos han tenido muchos nombres a lo largo de los años (DCOM, CORBA, EJB,
Servicios Web, etc.). Hoy los llamamos ``microservicios".

Python para Ciencia de Datos

En [3], se explica cómo el desarrollo de aplicaciones que involucran el aprendizaje automático requieren la exploración de los datos, el hallazgo de las variables correctas, la construcción y el ensayo de un modelo basado en un subconjuntos de datos. Una vez el modelo es encontrado y está funcionando correctamente es necesario desplegar el sistema sobre una plataforma para analizar los patrones de comportamiento de datos reales. En esta etapa el problema es diferente al que se tenía inicialmente, en este momento se tiene un entendimiento mucho más claro del problema y del modelo. El desafío en esta última etapa es diseñar un sistema que escale lo suficientemente bien para manejar a todos los usuarios y que sea lo suficientemente robusto para cambios futuros en los patrones de uso [2].
Python es uno de los lenguajes de programación más populares para manejar eficientemente los datos, tiene un excelente conjunto de librerías para la manipulación de información y para aprendizaje automático, este lenguaje es interpretado-dinámico, excelente para la exploración y el descubrimiento del modelo matemáticos que mejor se ajusten a los patrones de comportamiento de los datos. Por esta razón, sobre python se han desarrollado diversos *frameworks*.
Para la construcción de microservicios, en 2019, Gupta[1] [7] presenta la siguiente evaluación:
"...GIL es una limitación seria para las aplicaciones Python concurrentes vinculadas a la CPU, para las aplicaciones vinculadas a IO, la multitarea cooperativa de AsyncIO ofrece un buen rendimiento (más sobre esto más adelante). Para el rendimiento, deseamos un marco web que sea liviano pero maduro y que tenga API AsyncIO.".
Para la implementación de modelos apoyados en técnicas matemáticas de aprendizaje automático utilizando python, en 2020, Silaparasetty[1] [8] afirmó: "En comparación con otros lenguajes de programación, Python se destaca por las siguientes razones: * Funciona sin problemas en diferentes plataformas ... * Tiene una amplia lista de bibliotecas, utilidades y marcos específicos de aprendizaje automático, que los desarrolladores pueden usar para implementar aplicaciones en forma rápida y sencilla. Otros lenguajes no tienen bibliotecas tan extensas. Su código es simple, legible y conciso. * Por lo tanto, es más fácil

para un desarrollador centrar su atención en el problema de aprendizaje automático en cuestión, en lugar de tener que preocuparse demasiado por escribir el código correcto y siguiendo la sintaxis correcta, que requieren la mayoría de los otros lenguajes. * Tiene una gran comunidad de desarrolladores que utilizan con frecuencia el lenguaje. Esto hace que sea menos difícil para nosotros hacer preguntas sobre cualquier problema que surja mientras se programa.
Python tiene una gran variedad de bibliotecas de aprendizaje automático de código abierto, que incluyen, entre otras, las siguientes: * Pandas: la biblioteca Pandas brinda a los usuarios la capacidad de manejar grandes conjuntos de datos. * Numpy: la biblioteca Numpy, o Numerical Python, proporciona a los usuarios una poderosa variedad de habilidades informáticas. * Scipy: La biblioteca Scipy se utiliza para cálculos científicos y técnicos, trabaja con matrices multidimensionales de Numpy. * Scikit-Learn: la biblioteca Scikit-Learn consta de varias funciones y métodos que se han creado especialmente para ayudar a los usuarios en sus necesidades de aprendizaje automático. * TensorFlow: la biblioteca TensorFlow es una biblioteca cada vez más popular que proporciona a los usuarios un gran conjunto de herramientas flexibles y accesibles para el aprendizaje automático. ”

Metodología

Flujo de trabajo para un sistema de aprendizaje automático en línea

La figura 3, muestra las etapas que se siguieron para la implementación del sistema de aprendizaje automático para análisis de sentimientos.

Datos Históricos

La recolección de datos y preparación conlleva la captura de información, Brink[1] [3, p. 18] señala: “La recopilación y preparación de datos para los sistemas de aprendizaje automático generalmente implica obtener los datos en un formato tabular, si aún no lo está. Piense en el formato tabular como una hoja de cálculo en la que los datos se distribuyen en filas y columnas, cada fila corresponde a una instancia o ejemplo de interés, y cada columna representa una medición en esta instancia”.

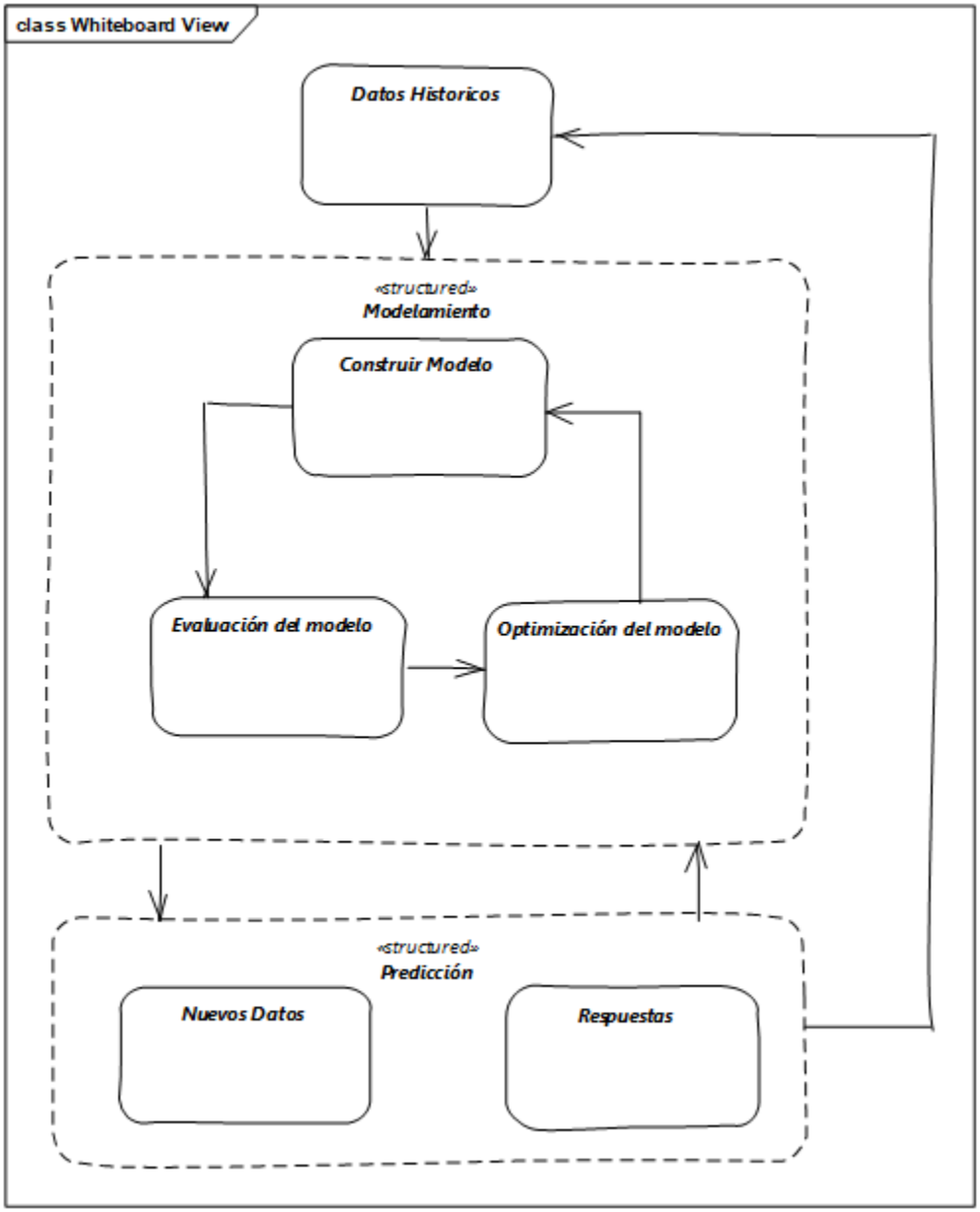


Figura 3 - Flujo de trabajo sistema aprendizaje de máquinas

Construcción del modelo

Brink[1] [3, p. 19] sobre la construcción del modelo dice: "La primera parte de la construcción de un sistema de aprendizaje automático exitoso es hacer una pregunta que pueda ser respondida por los datos ... piense en el algoritmo ML como una caja mágica que realiza el mapeo de las características de entrada a los datos de salida. Para crear un modelo útil, necesitaría más de dos filas. Una de las ventajas de los algoritmos de aprendizaje automático, en comparación con otros métodos ampliamente utilizados, es la capacidad de manejar muchas características.".
La creación del modelo para el análisis de sentimientos se hace en dos etapas:

La primera, el preprocesamiento de los datos, el cual consiste en obtener una representación de los textos en un espacio vectorial, esto se realiza usando la técnica conocida como bag of words, o matriz de términos en documentos. Esta representación se trabaja con bi-gramas.
La segunda, la implementación del modelo utilizando feed forward neural network con una arquitectura de 7 capas ocultas con 15, 50, 20, 50, 40, 40 y 20 neuronas respectivamente. El número de características corresponde al número de bigramas, 16133, para esta base de datos que es pequeña. La función de activación en todas las capas ocultas de la red es "ReLU" y sigmoidal para la capa de salida con una única neurona y la función de costo logística. El optimizador usado en el entrenamiento fue Adam y el número de épocas para el entrenamiento fue de 30. se usó cross validation con 10 folds en lugar de una partición train y test.

Evaluación del modelo

Para cada modelo que es puesto en producción se realizan validaciones de su desempeño, sobre este punto, en 2013 James[1] [11, p. 13] propuso: "necesitamos cuantificar hasta qué punto el valor de respuesta predicho para una observación dada se acerca al valor de respuesta real para esa observación. En la configuración de regresión, la medida más comúnmente utilizada es el error cuadrático medio (MSE), dado por $MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2$ donde $\hat{f}(x_i)$ es la predicción que da para la i-ésima observación. El MSE será pequeño si las respuestas predichas están muy cerca de las respuestas verdaderas, y será grande si para algunas de las observaciones, las respuestas predichas y verdaderas difieren sustancialmente." Por otro lado la metodología de validación que se utilizará en este proyecto es Cross-Validation, en 2006 Bishop[1] [12, p. 32] propuso "En muchas aplicaciones, sin embargo, el suministro de datos para entrenamiento y pruebas será limitado y, para construir buenos modelos, deseamos utilizar la mayor cantidad posible de datos disponibles para el entrenamiento. Sin embargo, si el conjunto de validación es pequeño, dará una estimación relativamente ruidosa del rendimiento predictivo. Una solución a este dilema es utilizar la validación cruzada. Esto permite una proporción (S −1)/S de los datos disponibles que se utilizarán para la formación y, al mismo tiempo, se utilizarán todos los datos para evaluar el rendimiento. Cuando los datos son particularmente escasos, puede ser apropiado considerar el caso S = N, donde N es el número total de puntos de datos, lo que proporciona la técnica de dejar uno fuera".
Se ejecutaron las siguientes actividades para alcanzar los objetivos planteados.

- Se utilizó una base de datos con textos que relataba la calidad de la atención en el servicio de hoteles, etiquetados como positivos o negativos.
- Documentación detallada de la arquitectura de software y de integración del sistema en AWS.
- Adaptación de la aplicación existente propuesto por el proyecto "Minería de Datos con Análisis de Sentimientos" para las fases de ingesta y entrenamiento del modelo.[2]
- Implementación del aplicativo "Sistema Reactivo para análisis de sentimientos con aprendizaje automático" siguiendo la arquitectura propuesta.
- Despliegue del sistema en la nube de AWS [21].

[2] http://grupotnt.udea.edu.co/sentiment-analysis

Arquitectura de Software

Un sistema con una arquitectura reactiva debe dar solución a cuatro principios fundamentales, Sensibilidad (Responsive), Elasticidad, Resiliencia y una comunicación entre componentes (microservicios) basada en mensajes [18].
La arquitectura que se propone en este trabajo para la ingesta, el entrenamiento y la puesta en producción de un sistema de aprendizaje de máquinas basado en análisis de sentimientos tiene como eje fundamental seguir los principios de un sistema reactivo.
La herramientas que se escogieron para la propuesta de arquitectura se basaron en los diferentes componentes de software proporcionados por Amazon Web Services - AWS [21].
AWS es una de las propuesta computación en la nube más completas del mercado la cual ofrece soluciones de cómputo, almacenamiento, y redes con diferentes niveles de abstracción.

Descripción de los servicios AWS utilizados

Amazon S3

Almacén de datos distribuido; S3 es el acrónimo de Amazon Simple Storage Service, es un servicio web que permite almacenar y tomar datos organizados como objetos vía una interfaz de programación de aplicaciones API, en inglés, application programming interface sobre el protocolo de transferencia de hipertexto seguro https [21].

SQS Queue

Es el acrónimo de Simple Queue Service. Servicio completamente administrado que permite crear colas de mensajerías que garantiza la entrega de los mensajes [22].

DynamoDB

Base de datos NoSQL del tipo almacén clave-valor con soporte para documentos. DynamoDB posee una alta disponibilidad y una larga duración, es posible escalar desde un item a millones y desde una solicitud por segundo hasta decenas de miles de solicitudes por segundo [23].

AWS Lambda

AWS Lambda proporciona un ambiente de ejecución para pequeñas funciones sin necesidad de preocuparse por el servidor que la contiene, este tipo de componente es llamado “serverless”, por que el servidor está oculto para el usuario y no es necesario preocuparse por la infraestructura subyacente [24].
Wittig [19, pp. 231] sobre serverless dice, se define un sistema serverless como uno que reúne los siguientes criterios:

- No es necesario administrar y mantener máquinas virtuales.
- Servicio completamente administrado que ofrece escalabilidad y alta disponibilidad.
- Se factura por solicitud y por consumo de recursos.
- Invoca una función para ejecutar el código en la nube.

Amazon API Gateway

Amazon API Gateway es un servicio administrado que facilita la creación, la publicación, el mantenimiento, el monitoreo y la protección de las API a cualquier escala. Las API AWS Gateway actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de *backend* [25].

Diagramas de Componentes y comunicación

Se presentan tres diagramas de arquitectura correspondientes a cada una de las fases por las que pasa un sistema de aprendizaje de máquinas, la ingesta de datos, el entrenamiento del modelo y la publicación del modelo.

Flujo de ingesta de datos

La figura 4 muestra la arquitectura de integración de la ingesta de datos, a continuación se describe cada una de las fases de la ingesta de datos.

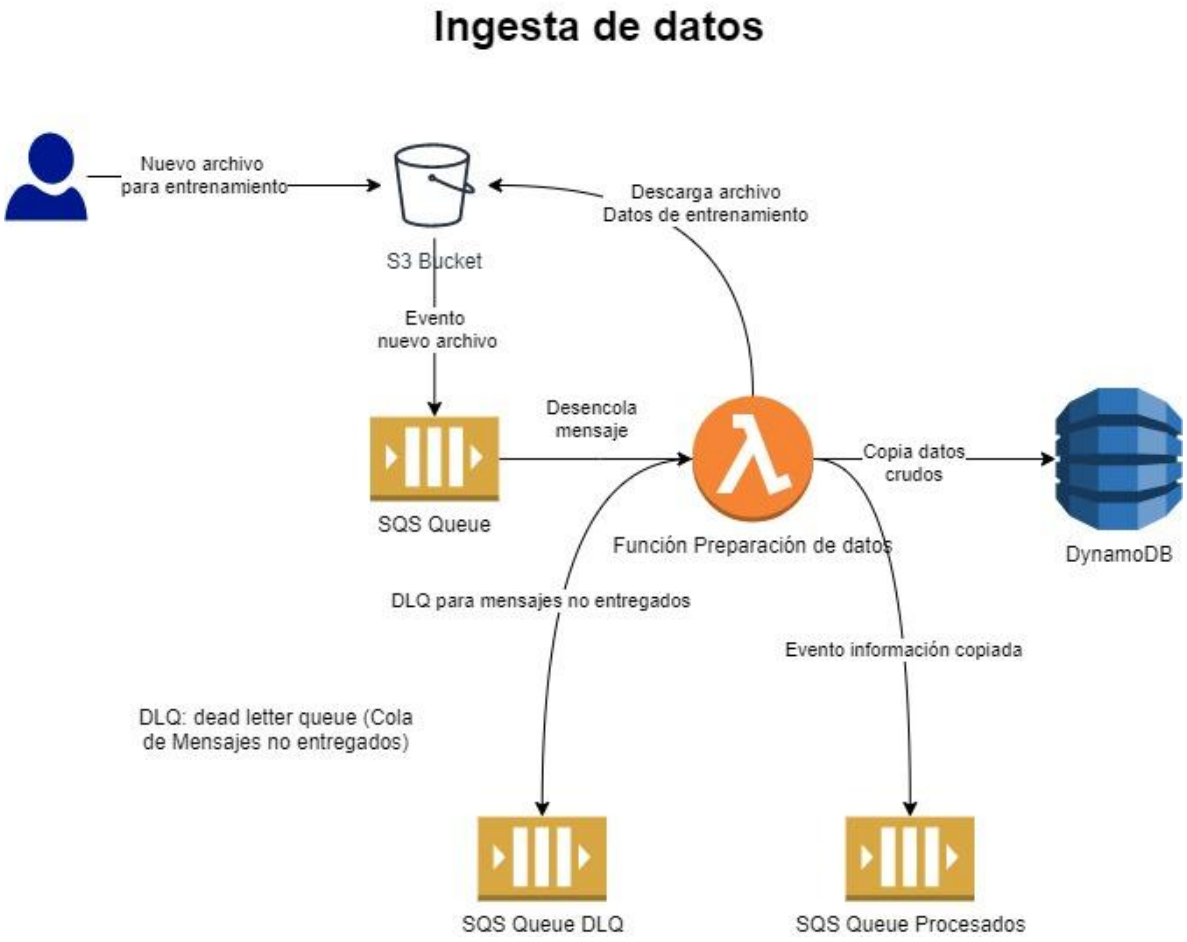


Figura 4 - Ingesta de datos

- Los usuarios con permisos sobre una carpeta de S3 designada para la recepción de información copian los archivos de texto que deseen con información de entrenamiento, ver figura 5.
- Una cola SQS recibe el evento de creación del archivo en la carpeta S3, y se almacena en la cola “SQS Queue” un mensaje con la ubicación del archivo recién copiado junto con información paramétrica adicional.

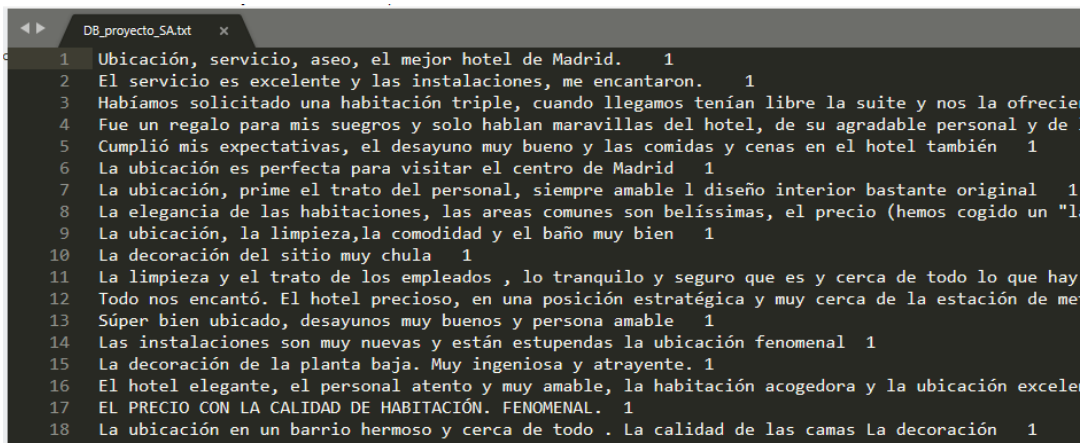


Figura 5 - Información base de entrenamiento

- Una función lambda asincrónica recibe en mensaje a través de la cola de mensajería "SQS Queue", con la información de la ubicación del archivo recién creada con esta información se verifica y lee el archivo copiado en la carpeta S3, se configuran tres intentos de lectura, si hay errores en la lectura del archivo se envía esta información a la cola "SQS Queue DLQ" en el que se almacenan los mensajes con problemas de procesamiento. Si la lectura del archivo es exitosa, se copia la información en una tabla clave-valor de DynamoDB y se envía un mensaje con la información de la tabla de DynamoDB de entrenamiento recién actualizada, ver figura 6.

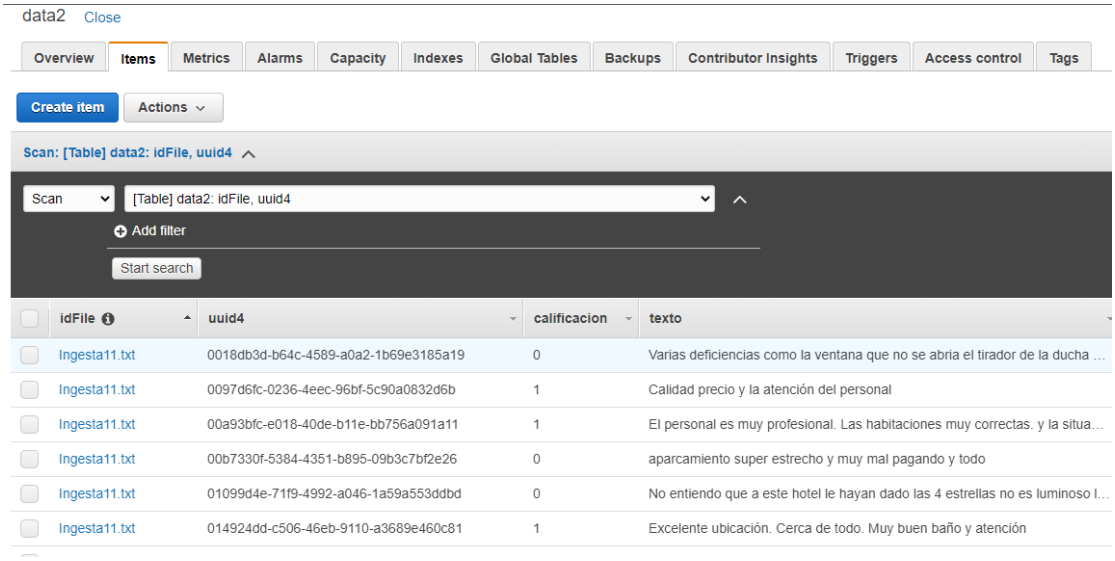


Figura 6 - Tabla data2

Flujo de entrenamiento del modelo

La figura 7 muestra la arquitectura de integración del entrenamiento del modelo, a continuación se describe cada una de las fases del entrenamiento.

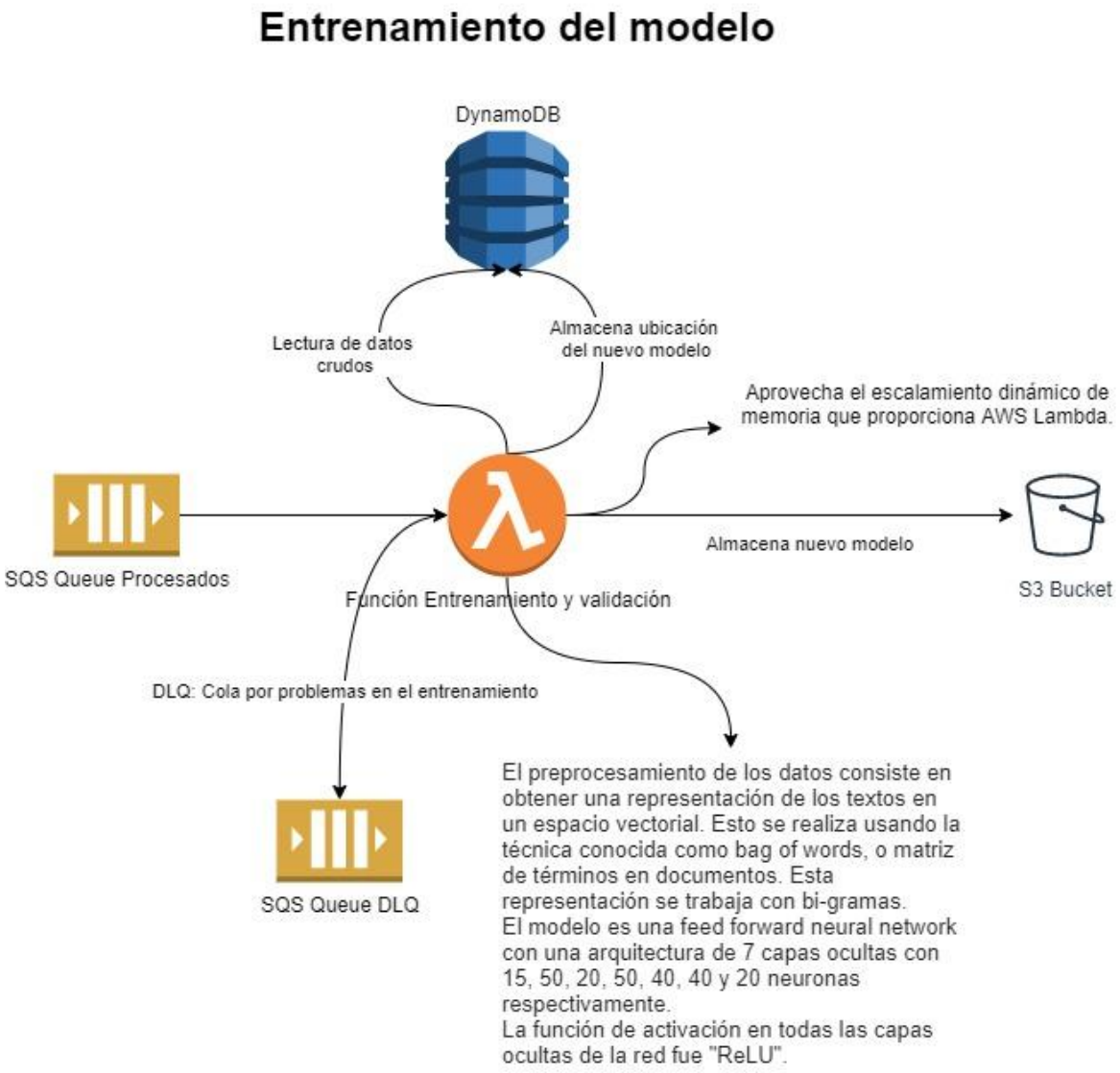


Figura 7 - Entrenamiento del modelo

- El mensaje de la cola SQS es leído por la función lambda de entrenamiento y validación.
- En el mensaje de la cola sqs indica el nombre del archivo recién cargado en s3 y el nombre de la tabla de DynamoDB en la que está almacenada la información precargada de entrenamiento.
- Con el nombre del archivo como clave se consulta desde la base de datos la información de entrenamiento.

- La función entrena el modelo con los datos recién suministrados utilizando una estrategia basada en una red neuronal de 7 capas y una función de activación en todas las capas ocultas de la red del tipo ReLu, utilizando Tensor Flow como librería de entrenamiento.
- Si el algoritmo de entrenamiento del modelo termina con éxito, el modelo es almacenado en una carpeta de s3, figura 8 (Se almacenan dos archivos el primero con extensión **joblib** y el segundo con extensión **h5** producto del entrenamiento), los nombre de los modelos, el nombre del archivo origen de entrenamiento y el nombre del bucket son registrados en una tabla de DynamoDB, figura 9.

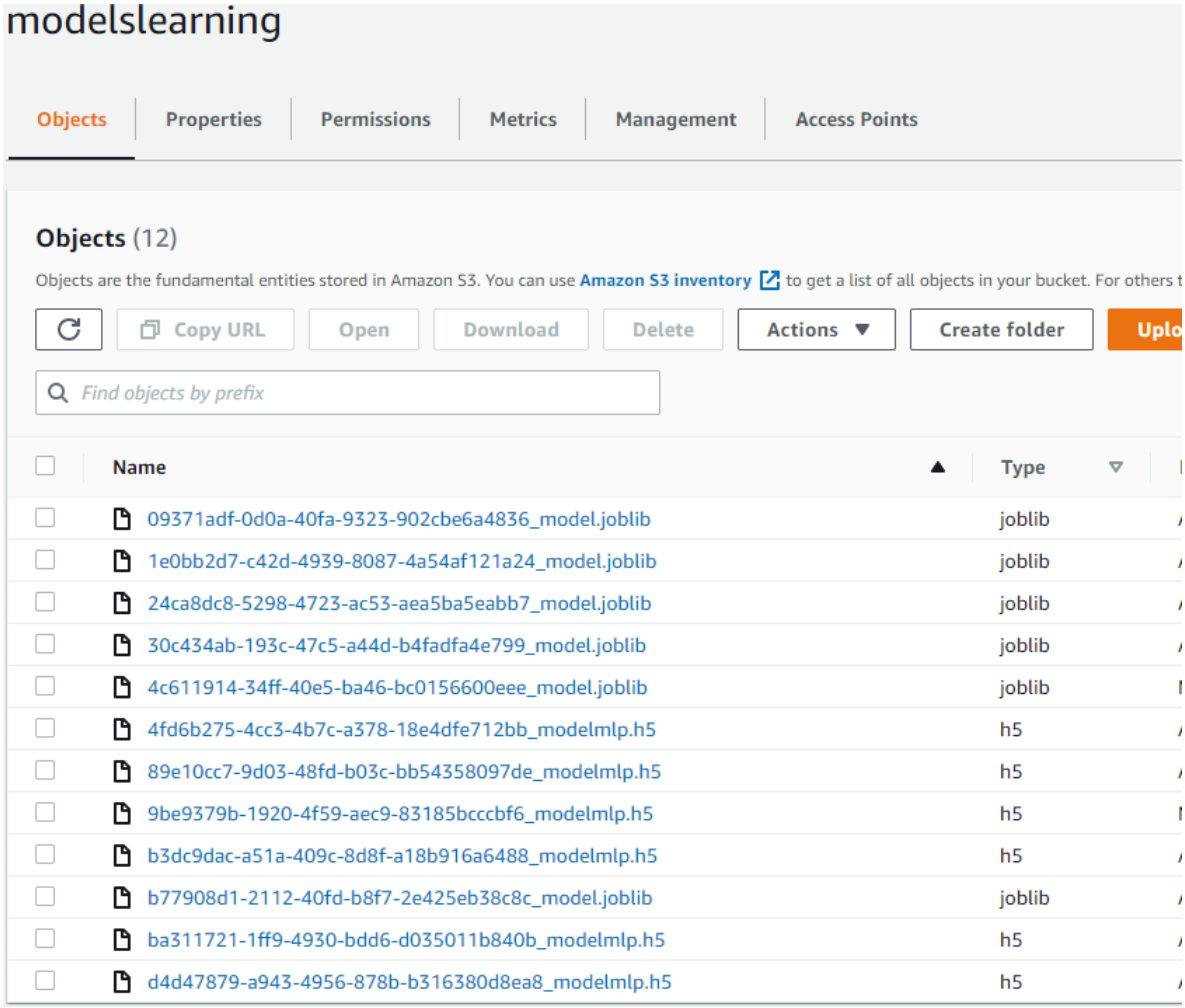


Figura 8 - Bucket de modelos entrenados

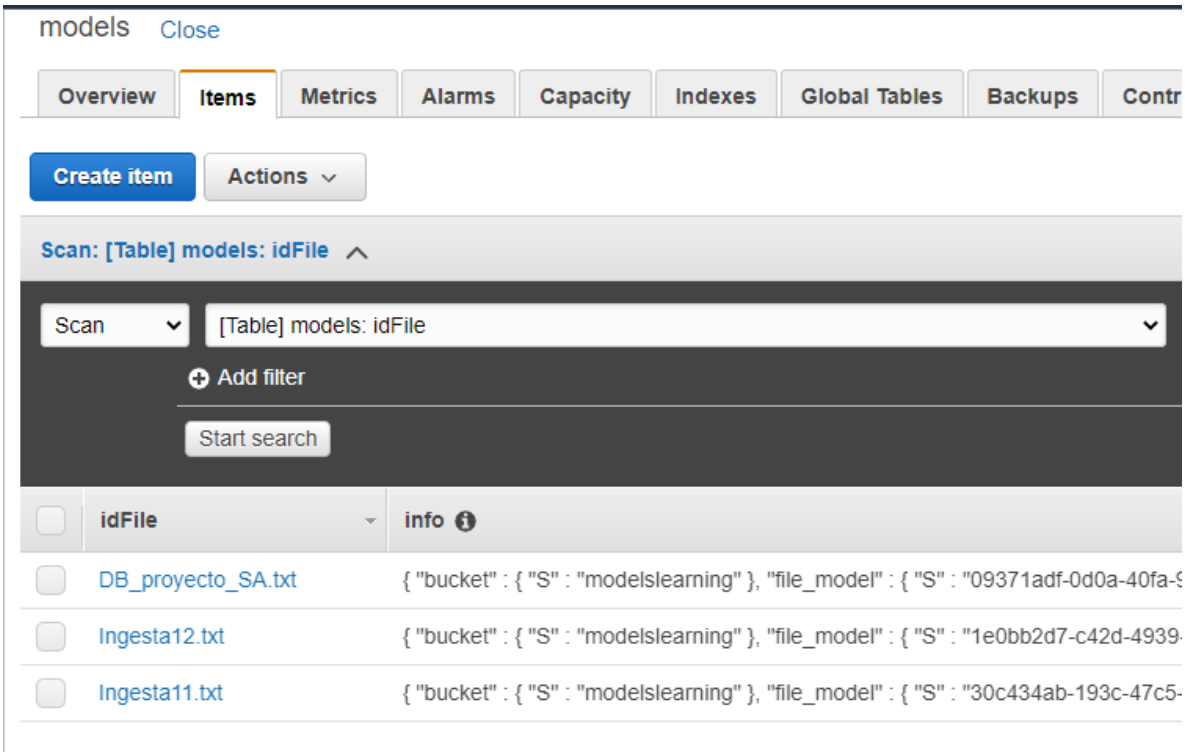


Figura 9 - Tabla models

- Si existe algún problema en el entrenamiento del modelo un mensaje es almacenado en una cola DLQ de SQS.

Flujo de publicación del modelo

La figura 10 muestra la arquitectura de la publicación del modelo.

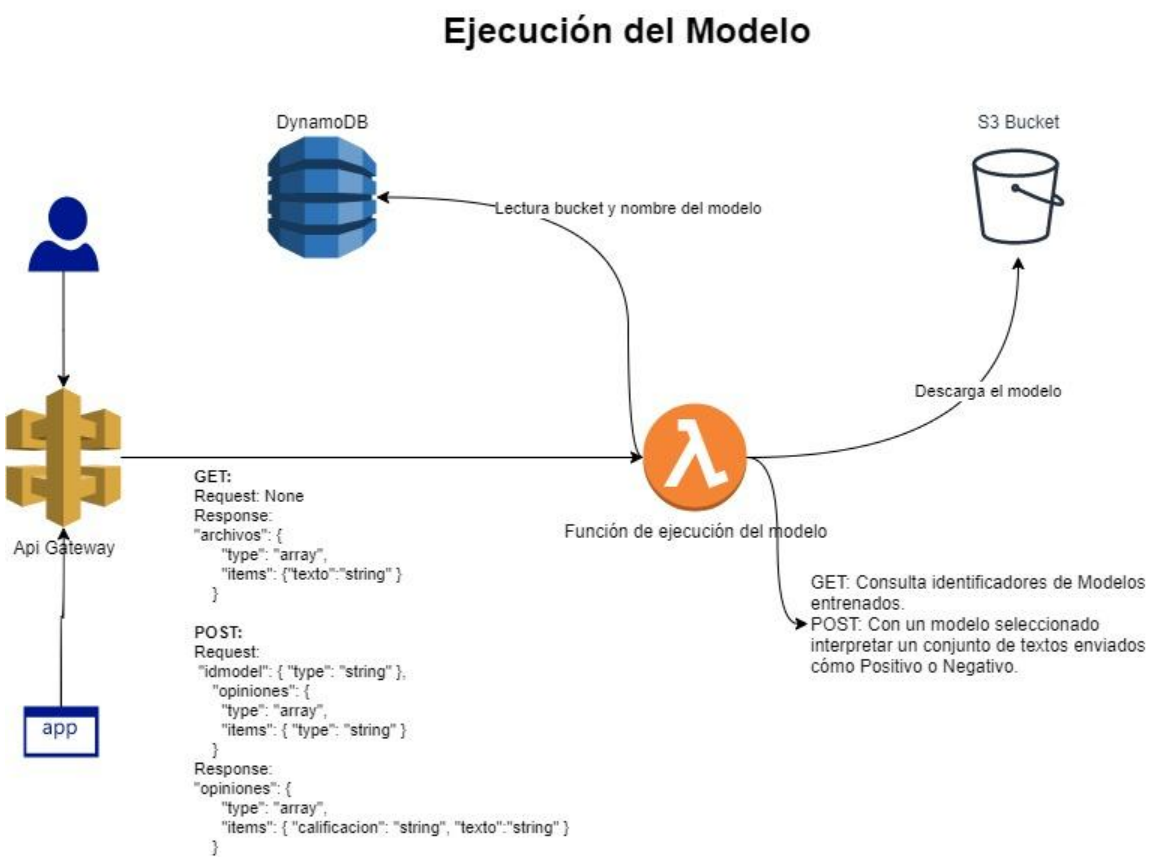


Figura 10 - Publicación del modelo

- El api gateway expone un servicio rest con los verbos get y post expuestos.
- El verbo get del api gateway se integra con la función lambda modelos que consulta los modelos entrenados.
  https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/modelos
- El verbo post del api gateway se integra con la función lambda de ejecución del modelo el cual recibe desde el cliente un identificador del modelo y una colección de cadenas de texto, las llamadas a este endpoint son enrutadas hacia la función lambda opinion.
  https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/sent
- La función de ejecución del modelo lambda opinion recibe un identificador del modelo y una colección de textos en español y devuelve cómo resultado del análisis del texto con valor "Positivo" o "Negativo" para cada texto, si estos expresan una opinión positiva o negativa respectivamente.
  Request:
  ```
  {
      "idmodel": "Ingesta12.txt",
      "opiniones": ["Lenovo Thinkpad es una excelente máquina, la recomiendo.",
      "Las camas estaban mal arregladas"]
  }
  ```
- La función lambda "opinion" descarga de S3 los archivos **h5 y joblib** asociados con el identificador del modelo.

- La función utiliza el modelo descargado y retorna el valor “positivo” o “negativo” por cada oración enviada dentro de la colección.

  Response:

```
[
        [
                "Positivo","Lenovo Thinkpad es una excelente máquina, la recomiendo."
        ],
        [
                "Negativo","Las camas estaban mal arregladas"
        ]
]
```

- Esta función lambda modelos expone un método sin parámetros de entrada y devuelve cómo resultado un listado de identificadores de modelos previamente entrenados consultando la tabla “models” de DynamoDB y retorna el listado de claves primarias que identifican los modelos.

  Response:

```
[
        {
                "idFile": "Ingesta11.txt"
        },
        {
                "idFile": "DB_proyecto_SA.txt"
        },
        {
                 "idFile": "Ingesta12.txt"
        }
]
```

Implementación

Creación de cuenta gratuita

Se crea una cuenta gratuita de AWS por 12 meses.

Creación de una identidad IAM en AWS

Se crea un usuario y se asigna al grupo Administrador, figura 11.

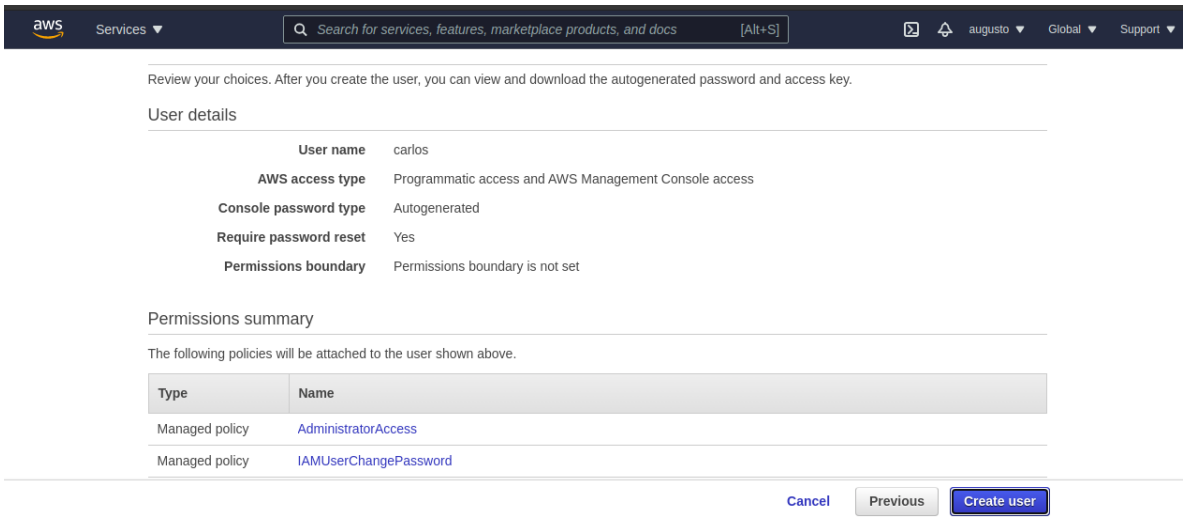


Figura 11 - Configuración Usuario

Ingesta de datos

Nombre función: **lambda-s3**

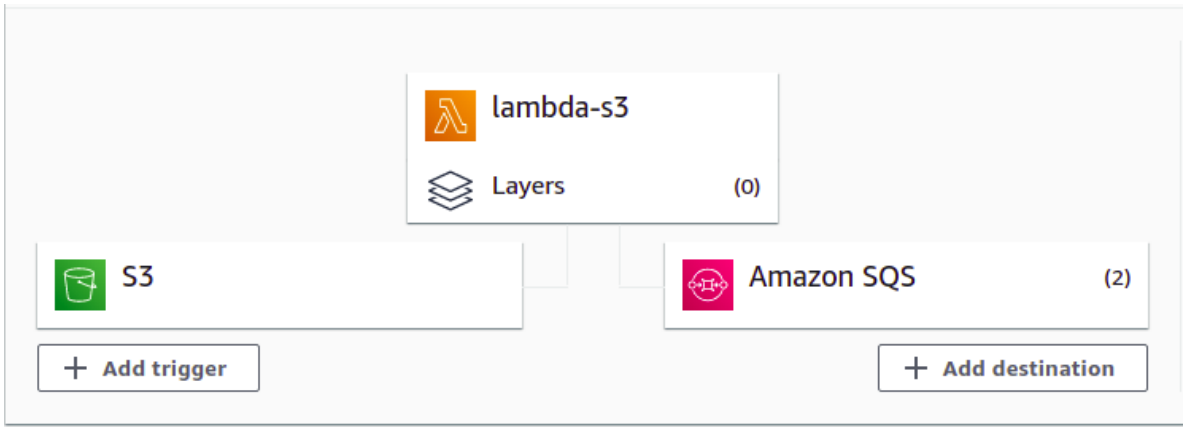


Figura 12 - Esquema lambda-s3

Trigger: S3

Event type: **ObjectCreated**

Notification name**: InvokeLambaOnNewObjects**

Prefix**: *None***

Suffix**: .txt**

Destino: Amazon SQS, figura 13

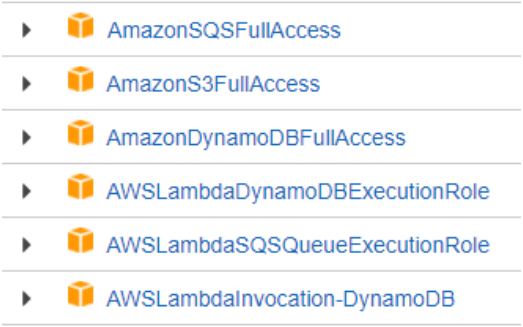
	Source	Stream	Condition	Destination
○	Asynchronous invocation	-	On success	arn:aws:sqs:us-east-2:722246141536:IngestaQueue
○	Asynchronous invocation	-	On failure	arn:aws:sqs:us-east-2:722246141536:dlqIngestaQueue

Figura 13 - Configuración destino SQS

Memoria: 2048 MB

Tiempo de espera: 40 s

Rol de ejecución: lambda-s3-role-w3vdsvmh
Políticas para lambda-s3-role-w3vdsvmh



Código:

Github: https://github.com/solrac68/training.git
Archivos: **training/source_training/src**/lambda_function.py
training/source_training/src/config.py

Comentarios

Esta función tiene la responsabilidad de leer el contenido de los archivos que son copiados en el bucket de s3 "files-training" y volcar la información en la tabla "data2" de DynamoDB.

El punto de inicio de la función es el método "lambda_function.lambda_handler", figura 14

```
52  def lambda_handler(event, context):
53      listaUbicacionesLocales = downloadFromBucket(event)
54      print(listaUbicacionesLocales)
55      data = [readFile(path,file) for path,file in listaUbicacionesLocales]
56      copyToDynamoDB(data)
57      return {
58          'statusCode': 200,
59          'body': {
60              'tablaDynamodb': config._TABLE_INGEST
61          }
62      }
```

Figura 14 - Función de inicio Lambda-S3

Este método divide el proceso en tres partes:

downloadFromBucket(event): Descarga de cada archivo dentro de la instancia de aws lambda. El parámetro de entrada event contiene el mensaje json con la información de la ubicación del archivo dentro en S3.

readFile(path,file): Lectura de cada archivo desde la máquina local en donde reside el proceso de ejecución de la función lambda.

copyToDynamoDB(data): Copia de la información de cada archivo en la tabla "data2" de DynamoDB.

El api que se utiliza en AWS para interactuar con los servicios de aws desde python es boto3 [20]. Cuando se crea una función lambda con python 3.8.3 cómo lenguaje de implementación el entorno de desarrollo en línea ya tiene pre-instalados estos paquetes y no

es necesario un archivo requirements.txt con la referencia a estas librerías para instalar la función.

Detalles del despliegue

Durante las primeras pruebas se observó que la función volcaba la información de forma parcial en la tabla “data2” de DynamoDB, y se producían re-intentos sucesivos de la función. Después de varios ensayos y ajustes se descubre que el incidente se solucionaba proporcionando más memoria ram que la asignada inicialmente por defecto (128 MB), el valor de 2048 MB fue el valor finalmente asignado con el que todas pruebas con archivos hasta un máximo de 1002 filas tuvieron éxito.

Entrenamiento del modelo

Nombre función: **training**

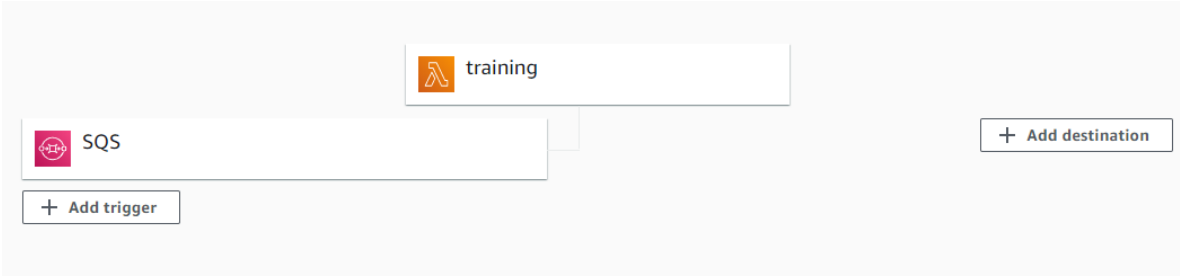
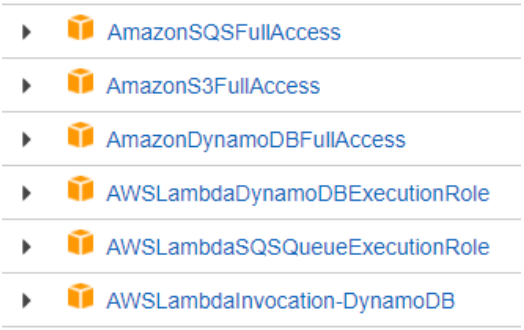


Figura 15 - esquema training

Trigger: SQS
Event type: **SQS**
Nombre Queue**: IngestaQueue**
Batch size: **10**
Prefix**:** ***None***

Destino: No configurado
Memoria: 3020 MB
Tiempo de espera: 5 min
Rol de ejecución: lambda-s3-role-w3vdsvmh
Políticas para lambda-s3-role-w3vdsvmh



Código:
Github: https://github.com/solrac68/training.git
Archivos: **training/source_training**
training/source_training/src/training.py
Comentarios
Esta función tiene la responsabilidad de leer la información cruda subida a DynamoDB en la tabla "datos2 utilizando como filtro la clave leída del mensaje de la cola **IngestaQueue**. Una vez descargada la información se lanza el proceso de entrenamiento utilizando las apis de Tensorflow y keras, aplicando un algoritmo de red neuronal de siete capas y generando como salida un archivo imagen comprimido de extensión joblib con la información de entrenamiento ("bag of words"), y un archivo del tipo h5 que almacena información estructurada correspondiente al modelo entrenado.
Los archivos joblib y h5 son subidos al bucket de S3 "modelslearning" para su posterior uso (Figura 9).
Los nombres y ubicación de los archivos son almacenados dentro de la tabla "models" de DynamoDb, figuras 16, 17 y 18.

```
params = {
    'TableName': tablaName,
    'KeySchema': [
        {'AttributeName': 'idFile', 'KeyType': 'HASH'}
    ],
    'AttributeDefinitions': [
        {'AttributeName': 'idFile', 'AttributeType': 'S'}
    ],
    'ProvisionedThroughput': {
        'ReadCapacityUnits': 10,
        'WriteCapacityUnits': 10
    }
}
```

Figura 16 - Estructura tabla models

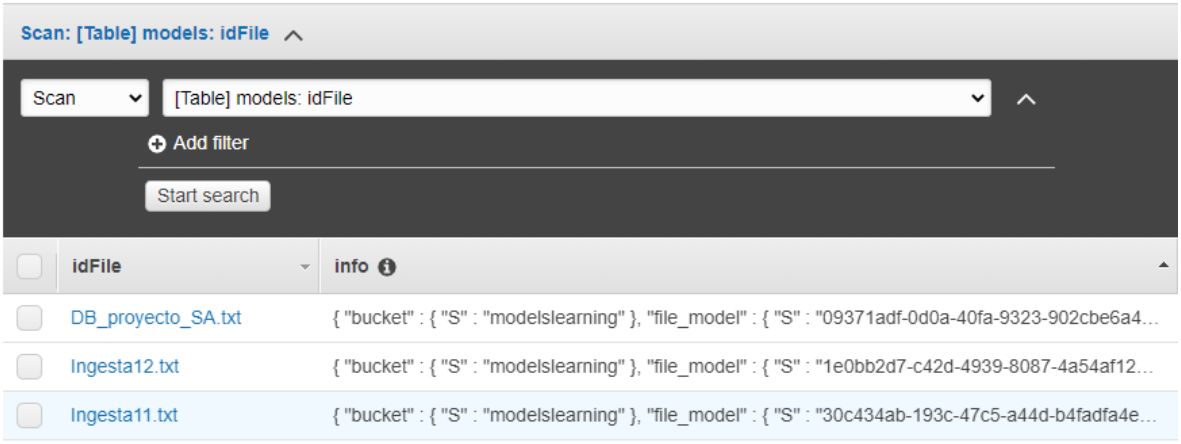


Figura 17 - Pantalla tabla models

```
▼ Item {2}
    idFile String : DB_proyecto_SA.txt
  ▼ info Map {3}
        bucket String : modelslearning
        file_model String : 09371adf-0d0a-40fa-9323-902cbe6a4836_model.joblib
        file_model_mlp String : ba311721-1ff9-4930-bdd6-d035011b840b_modelmlp.h5
```

Figura 18 - Registro de ejemplo

El punto de inicio de la función es el método “handler” del script training.py, figura 19.

```
260   def handler(event, context):
261       (nombreTabla, keys) = main_handler2(event, context)
262
263       return main_handler1(nombreTabla, keys)
```

Figura 19 - Función de inicio training

Este método divide el proceso en dos partes:
main_handler2(event, context): Toma el mensaje almacenado en la cola de mensajería dentro de event y saca la información del nombre de la tabla (nombreTabla) de DynamoDb en donde se encuentra la información de entrenamiento y las claves de búsqueda (keys).
main_handler1(nombreTabla, keys)*:* El método es responsable de lanzar el entrenamiento del modelo con la información cruda leída desde la tabla de DynamoDb.

Detalles del despliegue

Las librerías requeridas para el entrenamiento necesitan las librerías listadas en el archivo **training**/**source_training**/requirements.txt, figura 20.

```
1   boto3==1.17.53
2   tensorflow==2.4.1
3   keras==2.4.3
4   numpy
5   pandas==1.2.4
6   scikit-learn==0.24.1
7   joblib==1.0.1
8   matplotlib
```

Figura 20 - requirements.txt

Se creó una imagen de docker con python 3.8.3 y las librerías listadas en requirements.txt, al igual que la carpeta src con las fuentes del proyecto, figura 21.

```
FROM amazon/aws-lambda-python:3.8
RUN /var/lang/bin/python3.8 -m pip install --upgrade pip
RUN pip install --upgrade setuptools

##COPY##

COPY requirements.txt .
RUN pip install -r requirements.txt
COPY /src .

##ENTRYPOINT##
CMD [ "training.handler" ]
```

Figura 21 - Dockerfile

La imagen de docker es creada y etiquetada con los siguientes comandos:
$ sudo docker build -t lambda-docker-training
$ docker tag lambda-docker-training:latest 722246141536.dkr.ecr.us-east-2.amazonaws.com/lambda-docker-training:latest
La imagen se sube al contenedor de imágenes de AWS Elastic Container Registry - ECR [26] con los siguientes comandos:
$ aws ecr create-repository --repository-name lambda-docker-training --image-scanning-configuration scanOnPush=true
$ docker push 722246141536.dkr.ecr.us-east-2.amazonaws.com/lambda-docker-training:latest
Se verifica que la imagen fue desplegada a ECR dentro de la consola de administración de AWS, figura 21.

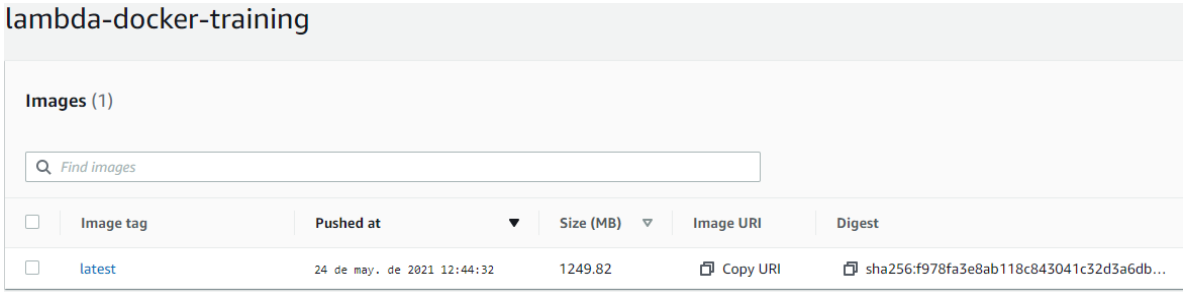


Figura 21 - imagen en ECR

Una vez la imagen se encuentra en ECR esta es referenciada desde la función AWS LAMBDA, figura 22.

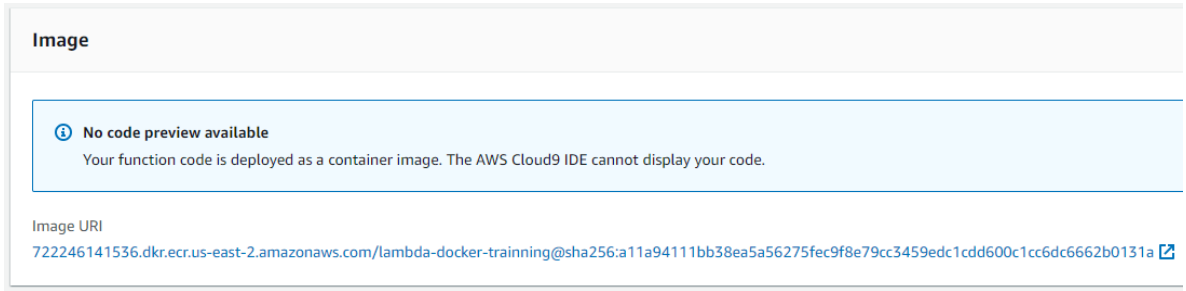


Figura 22 - Referencia de la imagen

Durante las primeras pruebas se observó que la función no terminaba correctamente los procesos, y se producían re-intentos sucesivos de la función. Se concluye que el incidente se soluciona proporcionando memoria ram adicional a la asignada inicialmente por defecto (128 MB), el valor de 3020 MB fue el valor finalmente asignado con el que todas pruebas con archivos hasta un máximo de 1002 filas tuvieron éxito.
En el proceso de entrenamiento del modelo TensorFlow emite una advertencia sobre la ausencia de gpus para el entrenamiento eficiente del modelo, sin embargo esta advertencia no evita que el proceso de entrenamiento se concluya con éxito.

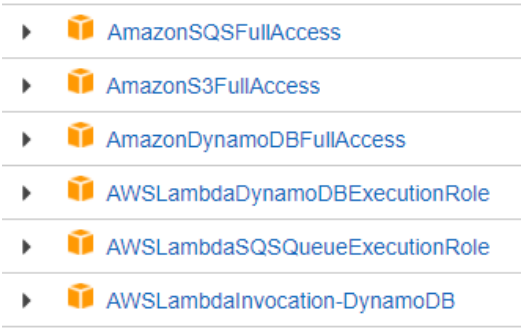
Publicación del modelo

Nombre función: **opinion**

Figura 14 - esquema opinion

Trigger: S3
Event type: **API Gateway**
API endpoint**:**
https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/senti
API type: **REST**
Authorization**: NONE**
Method**: POST**
Resource path**: /senti**

Memoria: 2048 MB
Tiempo de espera: 30 segundos
Rol de ejecución: lambda-s3-role-w3vdsvmh
Políticas para lambda-s3-role-w3vdsvmh



Código:
Github: https://github.com/solrac68/training.git
Archivos: **training/source_opinion/src/**opinion.py
training/source_opinion/src/config.py
training/source_opinion/src/bucket.py
training/source_opinion/src/dto.py

Comentarios
Esta función tiene la responsabilidad de evaluar los textos enviados a través del api gateway como positivos y negativos utilizando uno de los modelos previamente entrenados y almacenados en el bucket “models” de S3.
El punto de inicio de la función es el método “opinion.handler”, figura 15.

```
102  def handler(event, context):
103      (idmodel, opiniones) = getDatos(event, context)
104      return opinionGen(idmodel,opiniones)
```

Figura 15 - función de inicio “opinion”

Este método divide el proceso en dos partes:
getDatos(event, context): Toma desde el mensaje event enviado la identificación del modelo y el listado de textos que se analizarán.
opinionGen(idmodel,opiniones):

- Con el identificador del modelo se consulta en la tabla “models” de DynamoDb la ubicación y el nombre del modelo almacenado en S3.
- Se descargan el modelo almacenado en S3 dentro de la máquina local en donde reside la función.
- Se utilizan los modelos para evaluar la lista de oraciones como positivos o negativos.

Detalles del despliegue
Las librerías requeridas para la ejecución del modelo se encuentran listadas en el archivo **training**/**source_opinion**/requirements.txt, figura 20.

```
1  boto3==1.17.53
2  tensorflow==2.4.1
3  joblib==1.0.1
4  scikit-learn==0.24.1
```

Figura 20 - requirements.txt de “opinion”

Se creó una imagen de docker con python 3.8.3 y las librerías listadas en requirements.txt, al igual que la carpeta src con las fuentes del proyecto, figura 21.

```
FROM amazon/aws-lambda-python:3.8
RUN /var/lang/bin/python3.8 -m pip install --upgrade pip
RUN pip install --upgrade setuptools

##COPY##

COPY requirements.txt .
RUN pip install -r requirements.txt
COPY /src .

##ENTRYPOINT##
CMD [ "opinion.handler" ]
```

Figura 21 - Dockerfile “opinion”

La imagen de docker es creada y etiquetada con los siguientes comandos:

$ docker build -t lambda-docker-opinion .
$ docker tag lambda-docker-opinion:latest 722246141536.dkr.ecr.us-east-2.amazonaws.com/lambda-docker-opinion:latest
La imagen se sube al contenedor de imágenes de AWS Elastic Container Registry - ECR [26] con los siguientes comandos.
$ aws ecr create-repository --repository-name lambda-docker-opinion --image-scanning-configuration scanOnPush=true
$ docker push 722246141536.dkr.ecr.us-east-2.amazonaws.com/lambda-docker-opinion:latest
Se verifica que la imagen fue desplegada a ECR dentro de la consola de administración de AWS, figura 22.

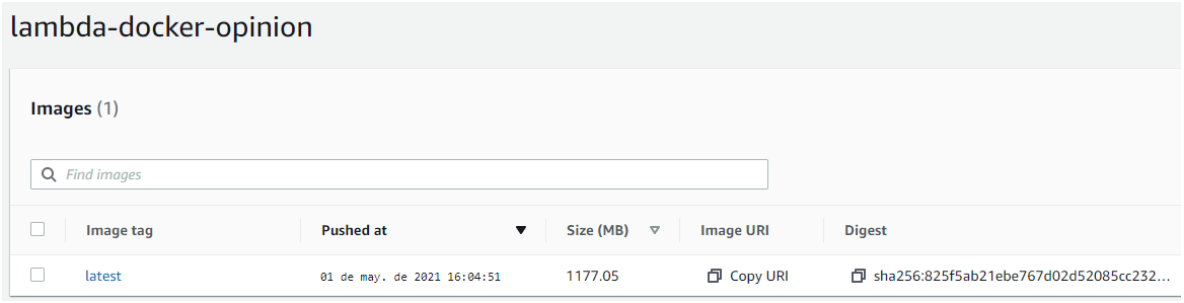


Figura 22 - imagen en ECR de "opinion"

Una vez la imagen se encuentra en ECR esta es referenciada desde la función AWS LAMBDA "opinion", figura 23.

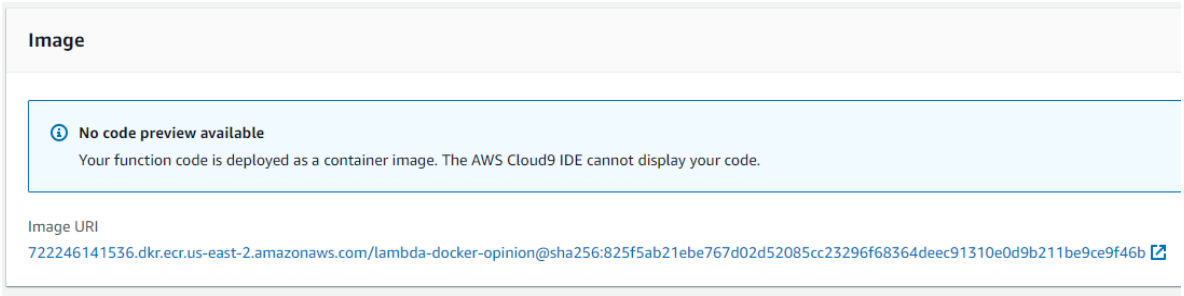


Figura 23 - Función de inicio training

Durante las pruebas se observó que la función no terminaba correctamente el proceso de análisis de textos, lanzando un error de time-out.

```
{
        "message": "Endpoint request timed out"
}
```

El problema se soluciono aumentando el tiempo de espera a un máximo de 30 segundos. Posteriores llamados a la función fueron exitosos con un tiempo de latencia máxima de un segundo.
Nombre función: **modelos**

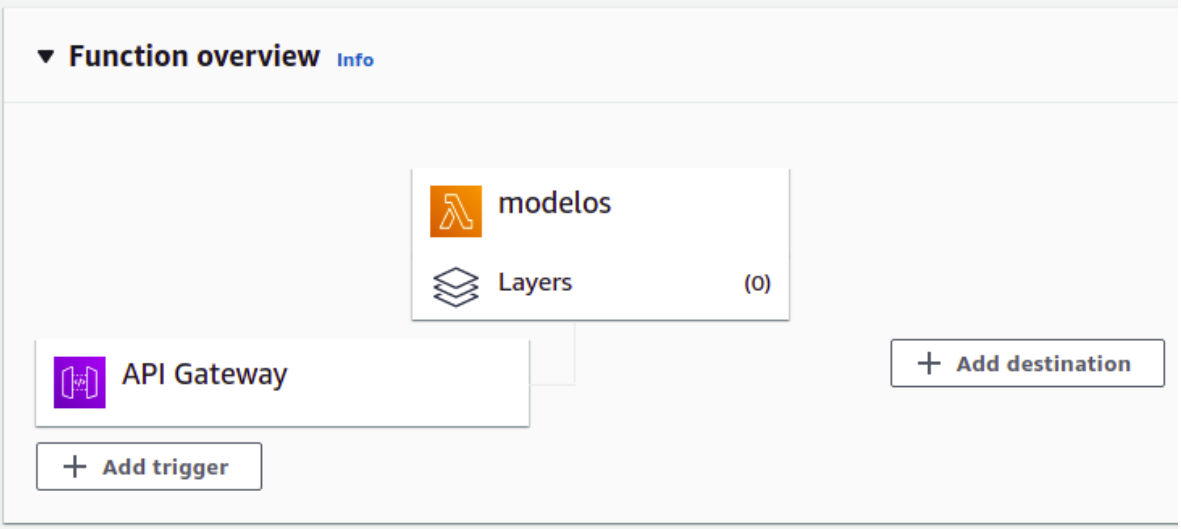


Figura 24 - esquema modelos

Trigger: S3

Event type: **API Gateway**
API endpoint**:** **https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/modelos**
API type: **REST**
Authorization**: NONE**
Method**: GET**
Resource path**: /modelos**

Memoria: 128 MB
Tiempo de espera: 3 segundos
Rol de ejecución: lambda-s3-role-w3vdsvmh
Políticas para lambda-s3-role-w3vdsvmh

- AmazonSQSFullAccess
- AmazonS3FullAccess
- AmazonDynamoDBFullAccess
- AWSLambdaDynamoDBExecutionRole
- AWSLambdaSQSQueueExecutionRole
- AWSLambdaInvocation-DynamoDB

Código:

Github: https://github.com/solrac68/training.git
Archivos: **training**/**source_opinion**/**src**/lambda_function.py
training/**source_opinion**/**src**/config.py

Comentarios
Esta función tiene la responsabilidad de consultar los identificadores de los modelos que fueron entrenados.

El punto de inicio de la función es el método “lambda_function.lambda_handler”, figura 25.

```
import config
import json
import dto

def lambda_handler(event, context):
    # TODO implement
    nombreTabla = config._TABLE_MODELS
    columna = "idFile"
    resultados = dto.queryKeys(nombreTabla, columna)

    jsonStr = json.dumps(resultados)

    return {
        "statusCode": 200,
        "headers":{
            'myHeader':'test'
        },
        "body": jsonStr,
        "isBase64Encoded": True
    }
```

Figura 25 - Función de inicio modelos

dto.queryKeys(nombreTabla, columna): Consulta el listado de nombres de los modelos desde la tabla “models” de dynamodb correspondiente a la columna “idFile”.
Detalles del despliegue
Esta función sólo hace uso de boto3 [20], el tiempo máximo de latencia presentada en la consulta es de 500 milisegundos.

Resultados y Análisis

Se muestra a continuación un caso de prueba dividido en ocho pasos con cada uno de los resultados que se obtienen producto del proceso de entrenamiento y ejecución del modelo.
Pasos 1 y 2 se observa el bucket “files-training” que recibe la información de entrenamiento del archivo DBA_PROYECTO_SA.txt, en el paso 3 se refiere a la tabla “data2” de DynamoDb que consolida la información depurada de entrenamiento producto del proceso de carga de información realizado por la función “lambda-s3”, en el paso 4 se presenta el log de resultados que entrega CloudWatch resultante del proceso de entrenamiento realizado por la función lambda “training” sobre los datos leídos desde la tabla “data2”, en el paso 5 se observa la ubicación de los modelos resultantes del entrenamiento almacenados en la tabla “models” de DynamoDB y en el paso 6 se verifican la creación de los modelos entrenados ubicados en el bucket “modelslearning”.
Para exponer los modelos para su ejecución se implementó un servicio api-gateway que se integra con dos funciones lambda, la primera es la función “opinion” y la segunda la función “modelos”.

En el paso 7, se muestra el llamado a la función “modelos” que retorna el listado de modelos entrenados por el sistema a través del cliente postman que invoca el verbo get de la url https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/modelos.
En el paso 8, se muestra el llamado a la función “opinion” la cual recibe un modelo de los previamente entrenados y una colección de oraciones, la función “opinion” utilizará el modelo seleccionado para interpretar cómo positivos o negativos cada una de las oraciones consultadas, la función fue invocada con la aplicación cliente postman utilizando el verbo post de la url https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/senti.
A continuación se describen en detalle cada uno de los pasos del escenario de prueba preparados para probar la implementación del sistema.

Paso 1

Acceder EL BUCKET “files-training” de s3, figura 26.

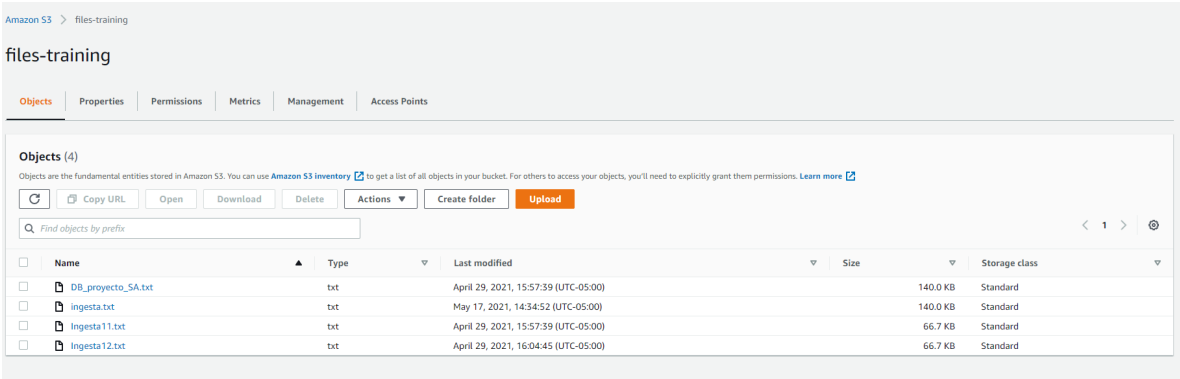


Figura 26 -Bucket files-training

Paso 2

Agregar archivo de entrenamiento, figura 27.

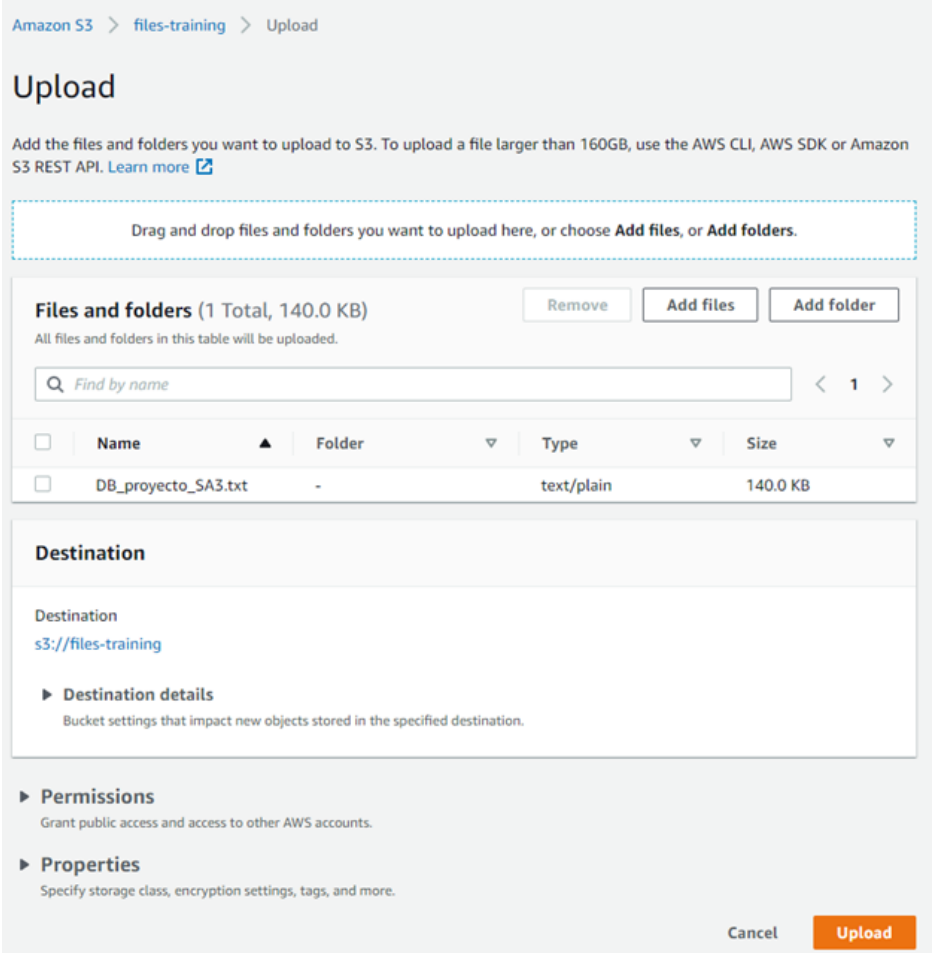


Figura 27 - Upload archivo

Paso 3

Verificar volcado de información cruda en la tabla data2 de DynamoDb, figura 28.

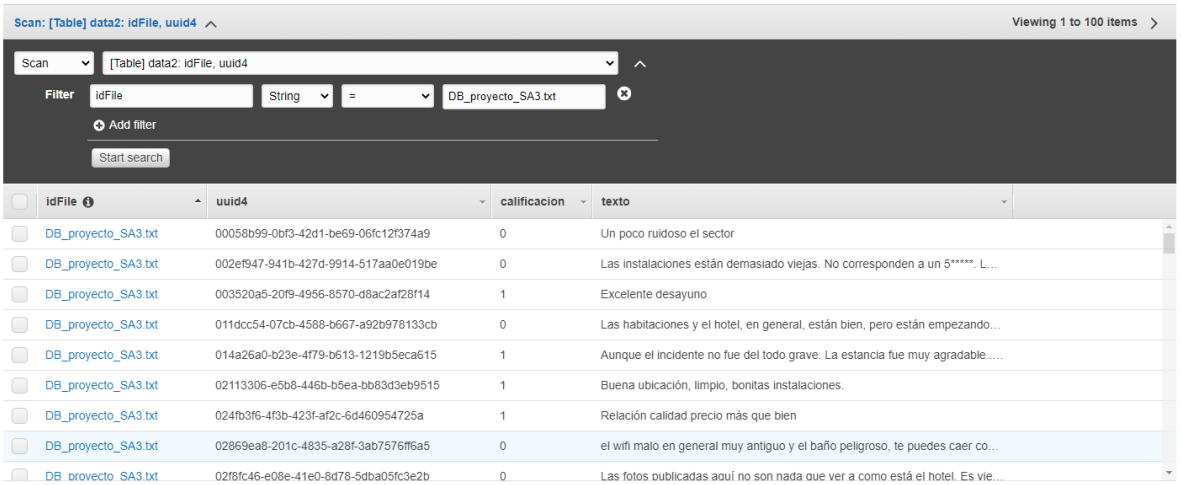


Figura 28 - Tabla data2 DynamoDb

Paso 4

Verificar los resultados del entrenamiento, función lambda “training” utilizando CloudWatch, figuras 29 y 30.

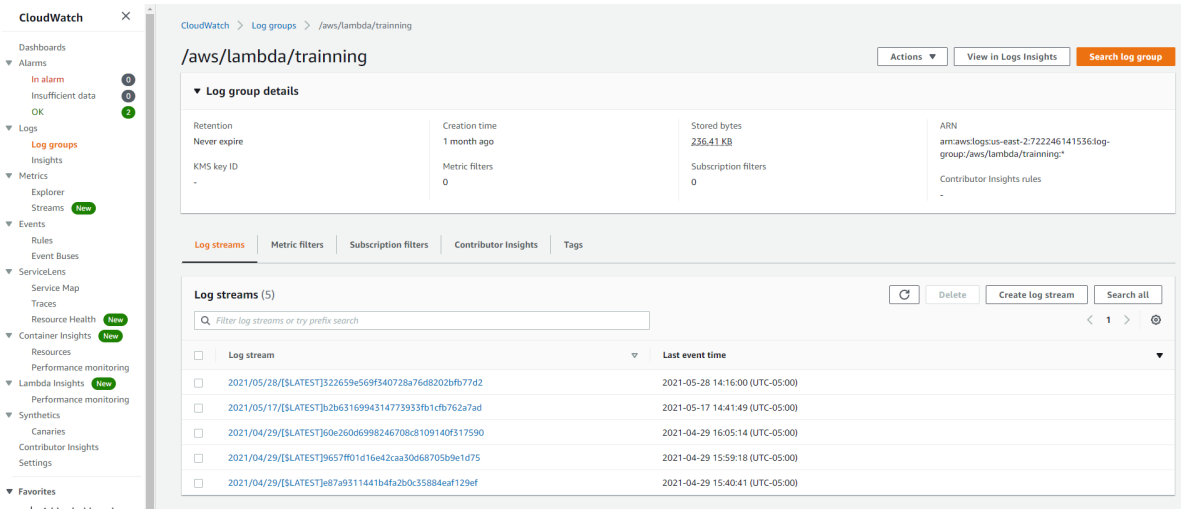


Figura 29 - CloudWatch

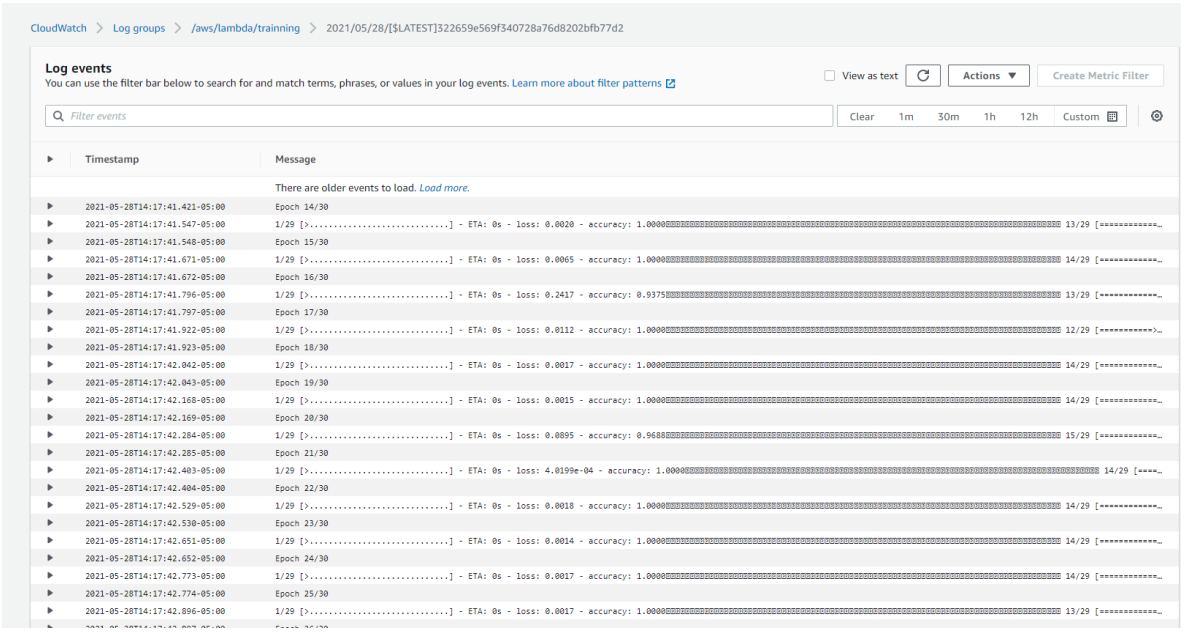


Figura 30 - CloudWatch Logs

En el log se puede evidenciar las siguiente salida al final del proceso:

Modelo e978b7bb-c057-4f6d-a5bc-3c3530bbc5fe_model.joblib creado en modelslearning S3
Modelo 08114548-1ca8-4b32-beb1-cbc6f0ed5eac_modelmlp.h5 creado en modelslearning S3

Modelos 08114548-1ca8-4b32-beb1-cbc6f0ed5eac_modelmlp.h5 y
e978b7bb-c057-4f6d-a5bc-3c3530bbc5fe_model.joblib registrados en tabla models
END RequestId: d8dd7173-522f-5a22-8aed-285811bb94e1
REPORT RequestId: d8dd7173-522f-5a22-8aed-285811bb94e1 Duration: 105481.41 ms Billed Duration: 105482 ms
Memory Size: 3020 MB Max Memory Used: 1619 MB

Paso 5

Verificar registro y ubicación de los modelos en la tabla “models” de DynamoDB, figuras 31 y 32.

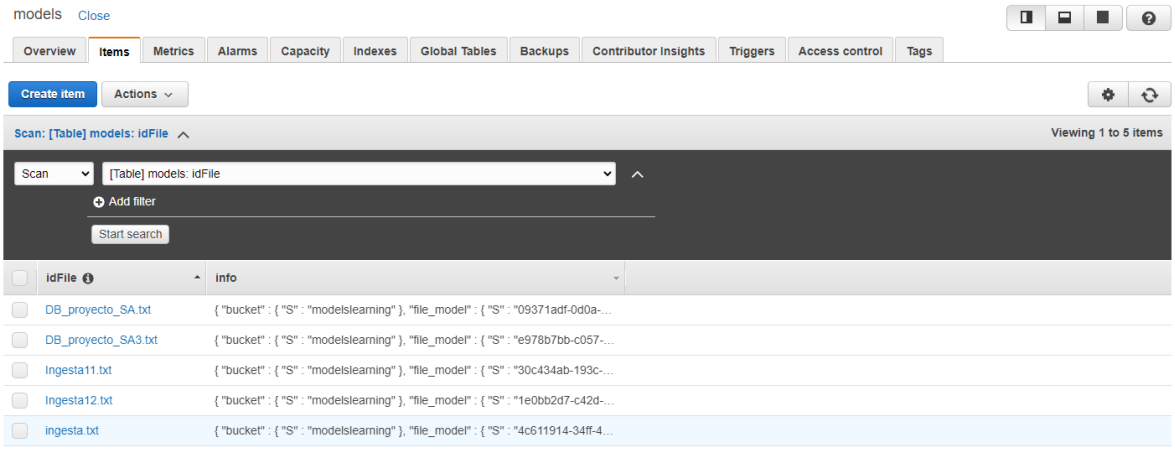


Figura 31 - Tabla models de DynamoDB

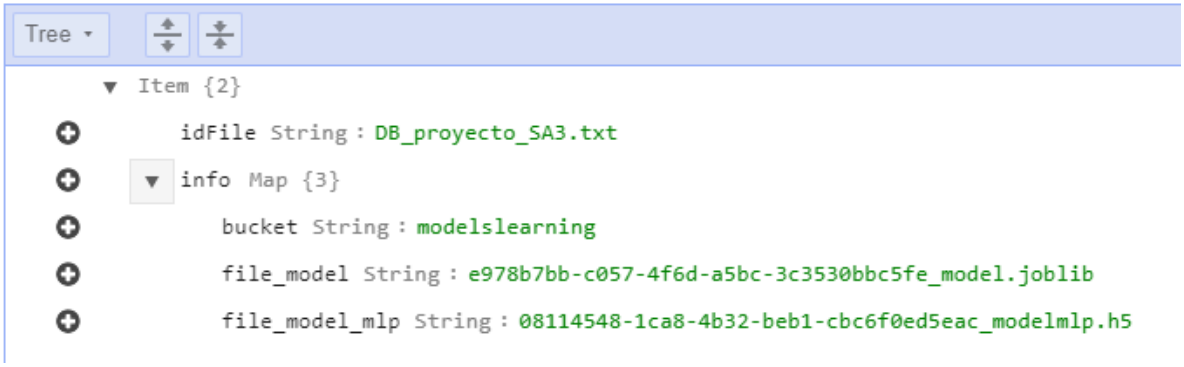


Figura 32 - Registro de la Tabla model

Paso 6

Verificar volcado de los modelos en el bucket “modelslearning” de S3, figura 33.

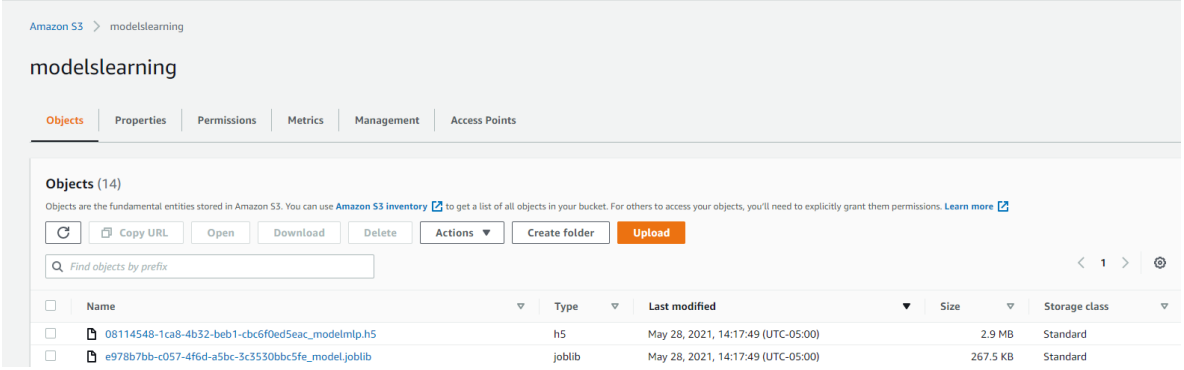


Figura 33 - Bucket modelslearning

Paso 7

Consultar los nombres de los modelos ya entrenados con el servicio REST utilizando el cliente para servicios Postman [31], figura 34.

GET

https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/modelos

RESPONSE


```
[
    {
        "idFile": "Ingesta11.txt"
    },
    {
        "idFile": "DB_proyecto_SA3.txt"
    },
    {
         "idFile": "ingesta.txt"
    },
    {
        "idFile": "DB_proyecto_SA.txt"
    },
    {
        "idFile": "Ingesta12.txt"
    }
]
```

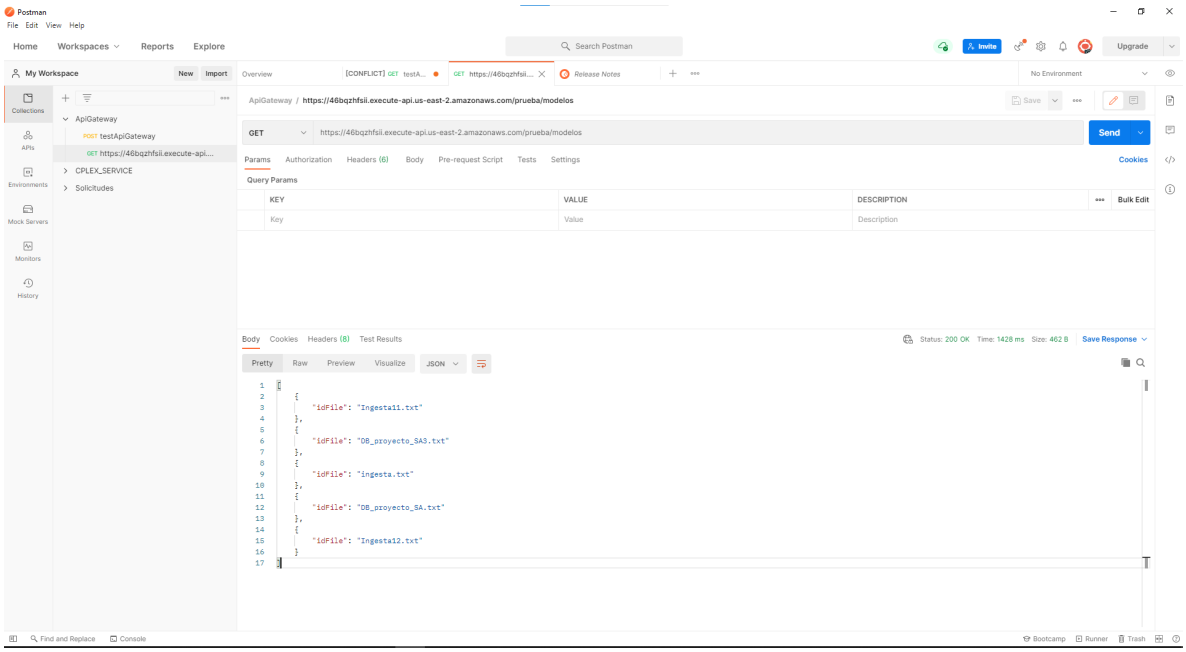


Figura 34 - get modelos

Paso 8

Utilizar el servicio rest para la evaluación de los textos utilizando uno de los modelos seleccionados utilizando el cliente para servicios Postman [31], figura 35.

POST

https://46bqzhfsii.execute-api.us-east-2.amazonaws.com/prueba/senti

```
{
        "idmodel": "DB_proyecto_SA3.txt",
        "opiniones": ["Lenovo Thinkpad es una excelente máquina, la recomiendo.", "Las camas estaban mal arregladas","El restaurante tiene comida muy mala", "Los camareros son personas muy amables","Los baños están sucios","El aire acondicionado no funciona bien en las mañanas"]
}
```

Response

```
[
  [
    "Positivo",
    "Lenovo Thinkpad es una excelente máquina, la recomiendo."
  ],
  [
    "Negativo",
    "Las camas estaban mal arregladas"
```

```
    ],
    [
        "Negativo",
        "El restaurante tiene comida muy mala"
    ],
    [
        "Positivo",
        "Los camareros son personas muy amables"
    ],
    [
        "Negativo",
        "Los baños están sucios"
    ],
    [
        "Negativo",
        "El aire acondicionado no funciona bien en las mañanas"
    ]
]
```

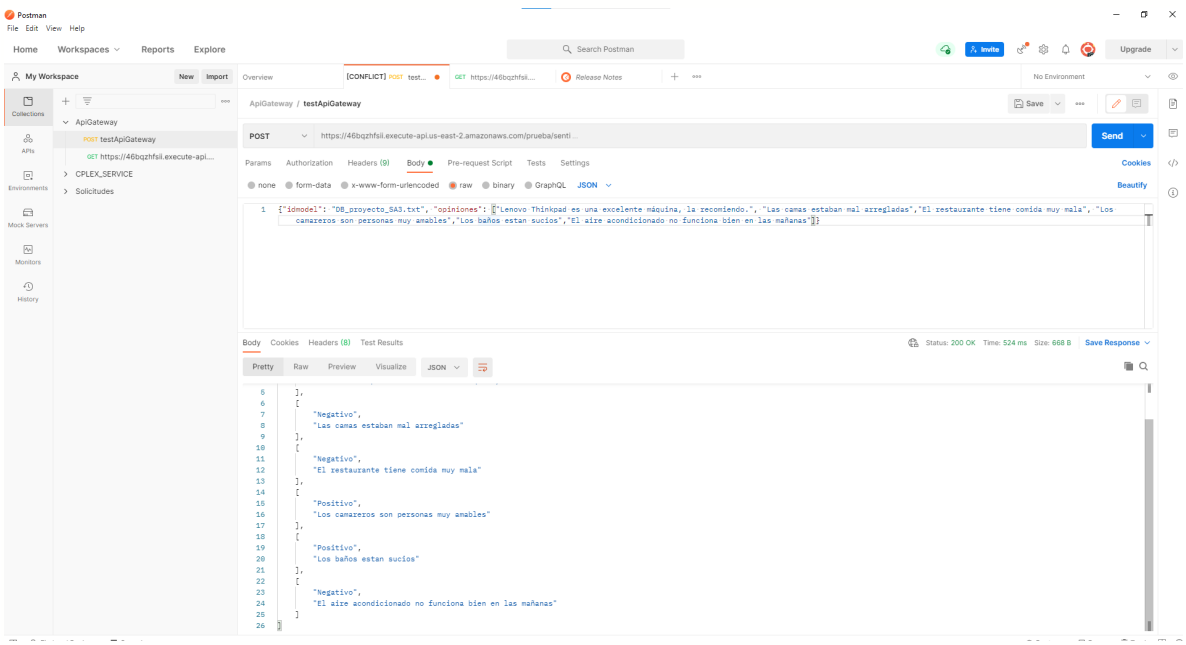



Figura 35 - post senti

En la Tabla 2 se presentan los principios de arquitecturas reactivas relacionados con los componentes del sistema de análisis de sentimientos implementado, cada principio está relacionado con un componente, la columna de comentarios justifica el por que cada uno de los componentes soporta el atributo no-funcional descrito por el principio.

Se resumen a continuación cada uno de los principios reactivos, Jonas [18] describe cada uno de los atributos de calidad reactivos de la siguiente forma:

Sensibilidad (Responsive): Un sistema es sensible (responsive) cuando este retorna constantemente una respuesta adecuada al usuario.

Resiliencia: El sistema se mantiene sensible (responsive) ante fallas, inclusive si se presenta una falla en el hardware o una falla humana.
Elasticidad: El sistema se mantiene sensible (responsive) con variaciones en la carga, significa que el sistema responde adecuadamente ante incrementos o decrementos en la carga.
Comunicación basada en mensajes: La comunicación entre los componentes que conforman el sistema es asincrónica, no bloqueante y basada en mensajes, en contraste con una comunicación tradicional entre procesos que genera una fuerte dependencia o acoplamiento entre las partes.
En la Tabla 3 presenta las estrategias que se aplicaron para alcanzar la implementación de la arquitectura reactiva, Jonas [18] describe cada una de las estrategias reactivas de la siguiente forma:
Replicación: Los componentes se ejecutan en más de un lugar al mismo tiempo.
Contención: Los componentes se aíslan en su ejecución para prevenir que la falla de un componente afecte otros componentes.
Supervisión: Se identifican los componentes que pueden fallar y explícitamente se asigna a otro componente la responsabilidad de monitorear su ciclo de vida. Esta estrategia da al sistema puntos de control que permitan reaccionar ante posibles fallas.

Principios	**Componente**	**Tipo**	**Comentario**
Sensibilidad (Responsive)	opinion	Microservicio	Retorna siempre una respuesta adecuada al usuario, utilizando modelos pre entrenados. Las funciones lambda permiten una concurrencia hasta de 3000 usuarios simultáneos, esto permite confiabilidad en la respuesta.
Resiliencia	training	Microservicio	Servicio asincrónico, al terminar cada entrenamiento informa a las colas de resultados el estado y permite reintentos. Las funciones lambda permiten un incremento de la capacidad de la memoria ram y la cuota de concurrencia de forma dinámica [27]. El sistema mantiene siempre un modelo

			entrenado, y aunque se presenten fallas en el entrenamiento previo, eso no afecta la respuesta de todo el sistema.
Resiliencia	opinion	Microservicio	Permite al usuario utilizar varios modelos entrenados e informa del estado. Las funciones lambda permiten un incremento de la capacidad de la memoria ram y la cuota de concurrencia de forma dinámica [27]
Comunicación basada en mensajes	Comunicación S3 -> LAMBDA-S3	Cola de mensajería SQS	Comunicación a través de una cola de mensajería implícita
Comunicación basada en mensajes	Comunicación LAMBDA-S3 -> trainnig IngestaQueue	Cola de mensajería SQS	Comunicación a través de una cola de mensajería
Elasticidad	training	Microservicio	Las funciones lambda permiten un incremento de la capacidad de la memoria ram y la cuota de concurrencia de forma dinámica [27]
Elasticidad	opinion	Microservicio	Las funciones lambda permiten un incremento de la capacidad de la memoria ram y la cuota de concurrencia de forma dinámica [27]

Tabla 2 - Principios reactivos

Estrategia	**Componente**	**Comentario**
Replicación	AWS lambda functions [24]	Todas las funciones lambda

		permiten aplicar estrategias de escalabilidad dinámica hasta alcanzar una cuota concurrente de 3000 instancias en regiones cómo Europa y EEUU este y Oeste [27]
Contención	Microservicios [5][6] AWS lambda functions [24] Elastic Container Registry - ECR [26] Docker [29]	Para prevenir que una falla en un componente del sistema afecte otros componentes se utilizó un diseño basado en microservicios utilizando imágenes de docker, registradas en ECR y desplegadas sobre AWS lambda.
Supervisión	Amazon CloudWatch [28]	CloudWatch es un componente de AWS que permite mantener supervisados todos los componentes que participan en el sistema, y reaccionar a potenciales fallas.

Tabla 3 - Estrategias Reactivas

Se seleccionó AWS [21] como plataforma en la nube sobre la que se implementó el sistema utilizando los servicios SQS [22], DynamoDB [23], AWS Lambda [24], Amazon API Gateway [25], Amazon Elastic Container Registry [26], Amazon CloudWatch [28] y para la construcción de algunos microservicios que lo ameritaba se utilizó Docker [29] como contenedor de los módulos de entrenamiento y análisis de oraciones.
El entrenamiento de los modelos fue hecho utilizando TensorFlow, Keras y un algoritmo conocido como bag of words sobre una red neuronal feed forward de siete capas ocultas con una función de activación ReLu. El entrenamiento proporcionó resultados satisfactorios para tamaños de ingesta de archivos de 1002 registros, en las que la función lambda tardó 90355 ms y una memoria ram utilizada de 1598 MB.
El punto crítico del sistema fue el entrenamiento del modelo, debido a que este proceso utilizó una mayor cantidad de recursos de cómputo. Se puede inferir en forma aproximada aplicando una regla de tres simple que la función de entrenamiento "training" tiene capacidad para procesar archivos de hasta de 1893 filas (3020/1002) * 1002.
Otro punto importante a resaltar es la advertencia que se presenta durante los procesos de entrenamiento y ejecución del modelo:
"*2021-05-17 19:40:36.944206: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.*"
Esta advertencia muestra que la máquina que se utiliza para la ejecución de las funciones

AWS-lambda encargadas del entrenamiento del modelo y de la ejecución no tiene instalados unidades de procesamiento gráfico (GPU), con unidades GPU el entrenamiento y la ejecución de los modelos se podrían realizar de forma mucho más óptima y en menor tiempo.

Trabajo futuro

La arquitectura presentada en este trabajo tiene limitaciones en cuanto a la capacidad para entrenar archivos mucho más grandes debido a las limitaciones que tiene las funciones lambda en cuanto al valor máximo de memoria ram (10240 MB) y a la capacidad de cómputo (no tienen unidades de procesamiento gráfico dentro el hardware responsable de la ejecución de las funciones).
Para subsanar esta limitación de la arquitectura se propone ampliar la arquitectura reactiva agregando AWS - Fargate [30] a la arquitectura, AWS - Fargate permite administrar cluster de contenedores y pagar solamente por los recursos necesarios para el cálculo.
En la figura 36, se propone una arquitectura que permitiría recibir archivos de entrenamiento mucho mayores y asignarle la responsabilidad de generación del modelo a una imagen docker del cluster aprovisionada y asociada a una máquina con GPU activo, AWS-Fargate gestionaría las imágenes, y cada imagen entregaría los resultados del procesamiento al bucket "modelslearning" de S3.

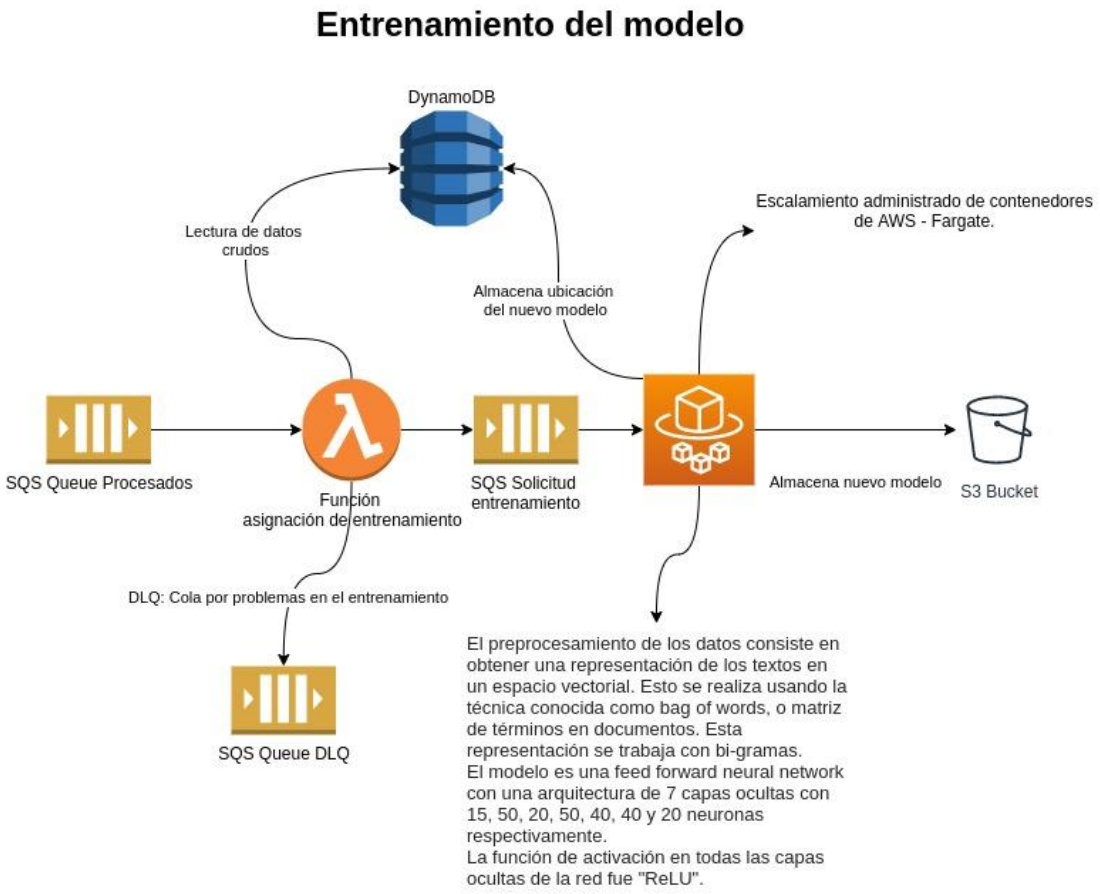


Figura 36 - Arquitectura entrenamiento del modelo propuesta

Conclusiones

Los conceptos de arquitecturas reactivas basadas en microservicios permiten solucionar los desafíos que los sistemas de aprendizaje automático presentan. En este trabajo se describió cómo se puede diseñar un sistema de aprendizaje de máquina para análisis de sentimientos creando microservicios reactivos para cada una de las fases por las que pasa el sistema, ingesta de los datos crudos, estructuración, limpieza, entrenamiento del modelo, y finalmente la publicación del modelo para su uso.

Se implementó un componente de entrenamiento de modelos para procesamiento de lenguaje natural aplicando la técnica conocida como bag of words junto con una red neuronal del tipo "feed forward" con una arquitectura de siete capas ocultas y una función de activación ReLu para todas las capas.

Las fases de ingesta de datos crudos, preparación, limpieza de los datos, carga de la información de entrenamiento, entrenamiento de los modelos y finalmente la publicación de los modelos fueron hechas utilizando servicios de AWS y siguiendo una arquitectura reactiva que se enfoca en los siguientes requisitos no-funcionales, sensibilidad (Responsive), resiliencia (Resilient), elasticidad (Elastic) y comunicación segura entre los componentes basada en mensajes (Message - Driven) utilizando las estrategias de replicación, contención y supervisión de componentes.

Se seleccionaron funciones aws-lambda [27] junto con el lenguaje de desarrollo Python 3.8 para la implementación de cada uno de los componentes del proyecto con un enfoque de microservicios sin servidor (serverless).

Para exposición del sistema en la nube de amazon se optó por un bucket de S3 [21] para la carga de archivos de entrenamiento y un servicio web rest expuesto con Amazon API Gateway [25] con dos operaciones, la primera para la consulta de los nombre de los modelos entrenados y la segunda que interpreta grupos de oraciones cómo positivas o negativas.

Una de las ventajas más relevantes que se encontraron al aplicar este tipo de arquitectura fue que el aislamiento de cada uno de los etapas del sistema de aprendizaje automático en componentes y la comunicación de estos a través de un sistema de cola de mensajería permite separar las responsabilidades asignadas a cada componente, esta estrategia evita un colapso completo del sistema al fallar uno de sus componentes, permite aplicar soluciones evolutivas a cada módulo, es decir, proponer soluciones sencillas inicialmente y posteriormente incrementar la complejidad a medida que se aumenta la carga sobre el sistema, sin afectar otros módulos que ya están en funcionamiento. En resumen, este tipo de solución permite un desacoplamiento funcional y físico de los cada uno de los microservicios que conforman el sistema y escalar la infraestructura que soporta cada módulo de forma independiente.

Referencias bibliográficas

[1] Bell J. (2015) Machine Learning, Hands-On for Developers and Technical Professionals, Indianapolis-IN-USA, John Wiley & Sons, Inc.

[2] Smith, J. (2018). Machine Learning System. N Y, EEUU: Manning Publications Co.

[3] Brink, H., Richard, JW., Fetherolf. (2017). M. Real World Machine Learning. N Y, EEUU: Manning Publications Co.

[4] McKee, H. (2016). *Designing Reactive Systems.* Sebastopol, CA, EEUU: O'Reilly

Media, Inc.
[5] Jonas Bonér. (2016). Reactive Microservices Architecture. Sebastopol,CA,USA:O'Reilly Media, Inc.
[6] Jonas Bonér. (2017). *Reactive Microsystems - The Evolution of Microservices at Scale.* Sebastopol, CA, USA: O'Reilly Media, Inc.
[7] Satish Chandra Gupta (2019), Python Microservices: Choices, Key Concepts, and Project setup,[online]. Available: https://medium.com/swlh/python-microservices-01-tornado-asyncio-lint-test-coverage-project-setup-9fbf4ca3bf90
[8] Nikita Silaparasetty. (2020) Machine Learning Concepts with Python and the Jupyter Notebook Environment - Using Tensorflow 2.0. Bangalore, India. Apress.
[9] Bugnion, P. (2016). *Scala for Data Science*. Birmingham B3 2PB, UK: Packt Publishing Ltd.
[10] Bruce Morgan, Pereira Paulo A (2019), Microservices in Action, Shelter Island, NY, USA, Manning.
[11] James, G., Witten, D., Hastie, T., Tibshirani, R. (2013) An Introduction to Statistical Learning with Applications in R. Stanford, CA, USA, Springer.
[12] Bishop M. C (2006), Pattern Recognition and Machine Learning, Cambridge - U.K, Springer.
[13] Leskovec J., Rajaraman A., Ullman J. (2014) Mining of Massive Datasets, Palo Alto, CA, Stanford.
[14] Singh P. (2019) Machine Learning with PySpark - With Natural Language Processing and Recommender Systems, Bangalore - Karnataka-India, Apress.
[15] BING L. (2015) Sentiment Analysis, New York - NY - USA, Cambridge University Press.
[16] KUHN ROLAND.,HANAFEE BRIAN., ALLEN JAMIE. (2017), Reactive Design Patterns, Shelter Island, NY, USA, Manning.
[17] Abhinav Ajitsaria (2019), What Is the Python Global Interpreter Lock (GIL)?, [online]. Available:https://realpython.com/python-gil/.
[18] Jonas Bonér, Dave Farley, Roland Kuhn, Martin Thompson. (2014, Septiembre 16), The reactive Manifesto (2014) [online]. Available: https://www.reactivemanifesto.org/
[19] Wittig Michael., Wittig Andreas. (2019), Amazon Web Services in Action, Second Edition, Shelter Island, NY, USA, Manning.
[20] Amazon Web Services, Inc. Created using Sphinx. (2021, Mayo 11) Boto3 Docs 1.17.70 documentation [online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html#
[21] Amazon Web Services (2021, Mayo 11) Explore nuestras soluciones [online]. Available: https://aws.amazon.com/es/s3/
[22] Amazon Web Services (2021, Mayo 11) Amazon Simple Queue Service [online]. Available: https://aws.amazon.com/es/sqs/
[23] Amazon Web Services (2021, Mayo 11) Amazon DynamoDB [online]. Available: https://aws.amazon.com/es/dynamodb/
[24] Amazon Web Services (2021, Mayo 11) AWS Lambda [online]. Available: https://aws.amazon.com/es/lambda/
[25] Amazon Web Services (2021, Mayo 11) Amazon API Gateway [online]. Available:

https://aws.amazon.com/es/api-gateway/?nc2=type_a

[26] Amazon Web Services (2021, Mayo 11) Amazon Elastic Container Registry [online]. Available: https://aws.amazon.com/es/ecr/

[27] Amazon Web Services (2021, Mayo 16) AWS Lambda function scaling [online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html

[28] Amazon Web Services (2021, Mayo 16) Amazon CloudWatch [online]. Available: https://aws.amazon.com/es/cloudwatch/

[29] Docker (2021, Mayo 16) Accelerate how you build, share and run modern applications [online]. Available: https://www.docker.com/

[30] Amazon Web Services (2021, Mayo 17) AWS Fargate - Cómputo sin servidor para contenedores [online]. Available: https://aws.amazon.com/es/fargate/

[31] The Collaboration Platform for API Development (2021, Junio 1) Postman [online]. Available: https://www.postman.com/