



Trabajo Fin de Máster

Máster en Ciencia de Datos e Ingeniería de Datos en la Nube

Análisis del sentimiento en las redes sociales

Autor: Tomás Pérez Campayo

Tutor: Juan Ignacio Alonso

Co-Tutor: ---

Mayo, 2021

*A mi pequeñajo... que por mi culpa
con tal solo 6 meses ya ha asistido
a más de una clase online sobre
Ciencia de Datos.*

Declaración de Autoría

Yo, Tomás Pérez Campayo con DNI 47077794Z, declaro que soy el único autor del trabajo fin de grado titulado Análisis del sentimiento en las redes sociales (COVID-19) y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 15 de Abril de 2021

Fdo: Tomás Pérez Campayo

Resumen

Las redes sociales hoy en día se han convertido en una fuente casi inagotable de información tanto para las empresas como para los gobiernos. Simplemente navegando por el perfil de cualquier usuario podemos conocer prácticamente cualquier faceta de su vida: sus gustos, sus aficiones, sus inclinaciones sexuales, políticas, religiosas, etc.

Gracias a la automatización de procesos de búsqueda, recolección y almacenamiento de datos, y aprovechando el número tan elevado de individuos que utilizan estas redes, podemos desarrollar sistemas capaces de reflejar la opinión o el sentimiento que despierta en la población un hecho o un acontecimiento dado. Al automatizar el análisis de sentimientos, se pueden procesar datos que, debido a su volumen, variedad y velocidad, no pueden ser gestionados de manera eficiente solo por seres humanos. Es imposible extraer el valor total de las interacciones en los centros de contacto, las conversaciones en las redes sociales, las reseñas de productos en foros y otros sitios web (en miles, si no cientos de miles) utilizando exclusivamente tareas manuales.

Durante prácticamente todo 2020 y veremos hasta cuando, el mundo entero se está viendo afectado por una pandemia que ya se ha cobrado en el mundo millones de vidas. La llegada del virus SARS-COV-2 ha trastocado la vida de toda la población mundial así como de muchas grandes y pequeñas empresas.

El objetivo de este proyecto puede resumirse en los siguientes puntos:

- Tomando como principal fuente de datos la red social **Twitter**, desarrollar un sistema que sea capaz de buscar, recolectar y almacenar tweets relacionados con el SARS-COV-2 así como de la enfermedad que provoca (COVID-19).
- Extraer el sentimiento (polaridad) de los tweets obtenidos y realizar una comparación con datos oficiales de contagios, hospitalizaciones, muertes, etc.

Agradecimientos

Primero de todo me gustaría agradecer a la empresa a la que orgullosamente pertenezco AIRBUS HELICOPTERS ESPAÑA el darme la oportunidad de seguir formándome y creciendo profesionalmente.

A mi mujer y especialmente a mi hijo, el mejor compañero de clase que uno puede tener.

Al resto de mi familia y amigos que siempre se preocupan e interesan por mí.

Y por supuesto a mi tutor, Juan Ignacio Alonso, cuya ayuda ha sido muy importante para realizar este proyecto.

Índice general

Capítulo 1	Introducción	1
1.1	Introducción	1
1.2	Objetivos	2
1.3	Estructura del proyecto	2
Capítulo 2	Estado del Arte	5
2.1	Introducción	5
2.2	Servicios en la nube: AWS	6
2.2.1	API GATEWAY	7
2.2.2	SNS	7
2.2.3	SQS	7
2.2.4	Funciones Lambda	8
2.2.5	DynamoDb	8
2.2.6	CloudWatch	8
2.2.7	Amazon Comprehend	9
2.3	Librería Tweepy	9
2.4	Serverless Framework	10
Capítulo 3	Metodología y Desarrollo	13
3.1	Introducción	14
3.2	Desarrollo en AWS	15
3.2.1	Lambda trigger_process	15
3.2.2	Lambda get_tweets	18

3.2.3	Lambda analyze_tweets	19
3.2.4	Lambda compute_statistics	23
3.3	Desarrollo con Serverless Framework	25
3.3.1	Instalación	25
3.3.2	Creación de un proyecto Serverless	25
3.3.3	Archivo serverless.yml	26
3.3.4	Código de las funciones lambda	32
3.3.5	Despliegue de la solución	33
3.3.6	Eliminación del servicio	34
3.4	Conclusiones	35
Capítulo 4	Experimentos y Resultados	36
4.1	Introducción	36
4.2	Servicio CloudWatch	36
4.3	Libreta Jupyter	38
Capítulo 5	Conclusiones y Trabajo Futuro	42
5.1	Conclusiones	42
5.2	Trabajo futuro	43
Bibliografía		44
Anexo I.	Recursos del proyecto	45
I.1	Recursos	45

Índice de figuras

Figura 2.1. Amazon Web Services	6
Figura 2.2. Tweeter logo.....	9
Figura 2.3. Serverless Framework	10
Figura 3.1. Arquitectura del sistema	14
Figura 3.2. Configuración básica de una función Lambda	16
Figura 3.3. Desencadenador ejecución función Lambda	17
Figura 3.4. Información del punto de enlace de llamada a la API	17
Figura 3.5. Asociación permisos Full Access al servicio SNS	17
Figura 3.6. Desencadenador SNS de la ejecución de la función Lambda	18
Figura 3.7. Configuración de permisos de acceso al servicio SQS	19
Figura 3.8. Desencadenador SQS de la ejecución de la función Lambda	19
Figura 3.9. Asociando permisos Full Access a la función Lambda	22
Figura 3.10. Asociación permisos Full Access sobre DynamoDB	24
Figura 3.11. Configuración archivo yml I	26
Figura 3.12. Configuración archivo yml II	27
Figura 3.13. Configuración archivo yml III	27
Figura 3.14. Configuración archivo yml IV	28
Figura 3.15. Configuración archivo yml V	29
Figura 3.16. Configuración archivo yml VI	30
Figura 3.17. Configuración archivo yml VII	30
Figura 3.18. Configuración archivo yml VIII	31
Figura 3.19. Configuración archivo yml IX	31
Figura 3.20. Configuración archivo yml X	32
Figura 3.21. Configuración archivo yml XI	32
Figura 3.22. Credenciales AWS.....	33
Figura 3.23. Informe de publicación del servicio en AWS	34
Figura 3.24. Confirmación de la eliminación de los servicios en AWS.....	35
Figura 4.1. Gráfica polaridad máximos por día.....	37

Figura 4.2. Gráfica polaridad mínimos por día	37
Figura 4.3. Gráfica polaridad valores medios por día	37
Figura 4.4. Muestra del set de datos	38
Figura 4.5. Valores totales por día	39
Figura 4.6. Hospitalizaciones, ingresos UCI y defunciones	39
Figura 4.7. Ejemplo contenido tabla tweets en Dynamodb	39
Figura 4.8. Hospitalizaciones, ingresos UCI y defunciones VS Polaridad	40
Figura 4.9. Muestra contenido tabla tweets_statistics en Dynamodb	40
Figura 4.10. Tweets positivos - negativos VS Polaridad obtenida	40

Índice de tablas

Tabla 3.1. Ejemplo registro tabla tweets en Dynamodb	22
Tabla 3.2. Ejemplo registro tabla tweets_statistics.....	24

Índice de código

Código 3.1. Publicación de un mensaje en el servicio SNS	16
Código 3.2. Llamada a la función search del API tweepy.....	18
Código 3.3. Envío de mensajes al servicio SQS	19
Código 3.4. Uso d la función detect_sentiment del API Comprehend	20
Código 3.5. Cálculo de los valores de sentimiento de un tweet concreto.....	20
Código 3.6. Añadiendo un registro a la tabla tweets.....	21
Código 3.7. Añadiendo un registro a CloudWatch.....	22
Código 3.8. Calculando valores acumulados por día	23
Código 3.9. Actualizando valores en la tabla tweets_statistics	24
Código 3.10. Comando de instalación Serverles Framework	25
Código 3.11. Comando de creación de un proyecto nuevo.....	25
Código 3.12. Comando de despliegue del servicio en AWS.....	33
Código 3.13. Comando de eliminación del servicio en AWS.....	35

Capítulo 1

Introducción

1.1 Introducción

El Análisis del Sentimiento es un área de investigación enmarcada dentro del campo del Procesamiento del Lenguaje Natural y cuyo objetivo fundamental es el tratamiento computacional de opiniones, sentimientos y subjetividad en textos. Podemos decir que una opinión es una valoración positiva o negativa sobre cualquier entidad que se nos ocurra (un producto, un servicio, una persona, etc).

Gracias a las redes sociales como Twitter se ha conseguido que este campo de investigación haya ido creciendo día a día de una forma más que sorprendente. Hay que destacar la importancia que supone el poder obtener la polaridad (grado de valoración) de miles de personas en un instante dado para los gobiernos, empresas y otras organizaciones.

Por otra parte, no podemos olvidar que el desarrollo de este tipo de sistemas ha sido posible gracias a la cantidad de información que tenemos accesible a través de Internet así como al aumento de la capacidad de cómputo del que disponemos gracias a los sistemas escalables en la nube.

1.2 Objetivos

El propósito de este proyecto es estudiar los sistemas de análisis del sentimiento a través de la implementación de uno que sea capaz de recabar información de la red social **Twitter** a cerca del virus SARS-COV-2 y la enfermedad COVID-19. Para ello se hará uso de varios servicios en la nube de AWS así como el lenguaje de programación Python.

Una vez desarrollado y desplegado el sistema, se programará una ejecución diaria en la que se recogerán 100 tweets, se analizará el sentimiento de cada uno de ellos y toda la información se almacenará en una base de datos.

Finalmente, con los datos obtenidos de la red social se realizará una comparación con datos sobre contagios, hospitalizaciones y muertes obtenidos de una fuente externa **Twitter**.

1.3 Estructura del proyecto

El proyecto se ha estructurado de la siguiente manera:

- La primera parte consiste en el desarrollo de un sistema basado en servicios AWS capaz de:
 - o Conectarse a la red social **Twitter** para extraer diariamente un conjunto de tweets relacionados con el SARS-COV-2.
 - o Procesar los tweets recogidos con el objetivo de obtener la polaridad (sentimiento) de cada uno de ellos.
 - o Almacenar los resultados en una base de datos y en un sistema de logging para su posterior análisis.
- La segunda parte consiste en replicar la arquitectura y comportamiento descrito en el primer punto haciendo uso de Serverless framework. Con el uso de esta tecnología veremos cómo podemos incrementar la productividad en el despliegue, pruebas y puesta en producción de un sistema en la nube.
- La tercera y última parte consiste en desarrollar una libreta de Jupyter en la que se muestra un pequeño análisis de los resultados obtenidos de la red social

Twitter cruzándolos a su vez con datos oficiales de número de contagios, hospitalizaciones y muertes por COVID-19.

Capítulo 2

Estado del Arte

2.1 Introducción

A continuación se van a explicar y detallar las tecnologías utilizadas para el desarrollo de este proyecto.

2.2 Servicios en la nube: AWS



Figura 2.1. Amazon Web Services

Amazon Web Services (AWS) es la plataforma en la nube más adoptada y completa en el mundo, que ofrece más de 200 servicios integrales de centros de datos a nivel global. Millones de clientes, incluso las empresas emergentes que crecen más rápido, las compañías más grandes y los organismos gubernamentales líderes, están usando AWS para reducir los costos, aumentar su agilidad e innovar de forma más rápida.

AWS ofrece desde tecnologías de infraestructura como cómputo, almacenamiento y bases de datos hasta tecnologías emergentes como aprendizaje automático e inteligencia artificial, Datalakes y análisis e internet de las cosas. Esto hace que llevar las aplicaciones existentes a la nube sea más rápido, fácil y rentable y permite crear casi cualquier cosa que se pueda imaginar.

AWS tiene la infraestructura en la nube más amplia del mundo. Ningún otro proveedor de servicios en la nube ofrece tantas regiones con varias zonas de disponibilidad conectadas por redes de baja latencia, alto rendimiento y alta redundancia. AWS cuenta con 80 zonas de disponibilidad repartidas en 25 regiones geográficas de todo el mundo. Además, se han anunciado planes para incorporar otras 15 zonas de disponibilidad y 5 regiones de AWS adicionales en Australia, India, Indonesia, España y Suiza. (Amazon Web Services, 2021)

A continuación se detallan los servicios utilizados para el desarrollo del proyecto:

2.2.1 API GATEWAY

Es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Las API 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. Se pueden crear API RESTful que están optimizadas para cargas de trabajo sin servidor y backends HTTP mediante API HTTP y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real. API Gateway admite cargas de trabajo en contenedores y sin servidor, así como aplicaciones web. (AWS - API Gateway, 2021)

2.2.2 SNS

Amazon Simple Notification Service (SNS) es un servicio de mensajería completamente administrado para la comunicación aplicación a aplicación (A2A) y aplicación a persona (A2P). Permite desacoplar las aplicaciones en componentes independientes y más pequeños que son más fáciles de desarrollar, implementar y mantener. Utilizar una arquitectura de este tipo basada en eventos de mensajería de publicación/suscripción mejora el rendimiento, la fiabilidad y permite escalar cada componente de forma independiente. (AWS - SNS, 2021)

2.2.3 SQS

Amazon Simple Queue Service (SQS) es un servicio de colas de mensajes completamente administrado que permite desacoplar y ajustar la escala de microservicios, sistemas distribuidos y aplicaciones sin servidor. SQS elimina la complejidad y los gastos generales asociados con la gestión y el funcionamiento del middleware orientado a mensajes, y permite a los desarrolladores centrarse en la diferenciación del trabajo. Con SQS, se pueden enviar, almacenar y recibir mensajes entre componentes de software de cualquier volumen, sin pérdida de mensajes ni la necesidad de que otros servicios estén disponibles. (AWS - SQS, 2021)

2.2.4 Funciones Lambda

AWS Lambda es un servicio que permite ejecutar código sin aprovisionar ni administrar servidores. Lambda ejecuta el código solo cuando es necesario y se escala de manera automática, pasando de pocas solicitudes al día a miles por segundo. Una de las ventajas de este servicio es que únicamente se paga por el tiempo de ejecución que se consume, es decir, no se aplican cargos cuando el código no se está ejecutando. Con este servicio se puede ejecutar código para prácticamente cualquier tipo de aplicación o servicio de backend, y sin que se requiera ningún tipo de administración.

Lambda ejecuta el código en una infraestructura de alta disponibilidad y ejecuta la administración integral de los recursos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como la monitorización y los registros.

Por otra parte, se puede utilizar Lambda para ejecutar el código en respuesta a eventos, como cambios en los datos en un Amazon Simple Storage Service (Amazon S3) bucket o una tabla Amazon DynamoDB o para ejecutar el código en respuesta a solicitudes HTTP utilizando Amazon API Gateway. También se puede utilizar Lambda para crear desencadenadores de procesamiento de datos para AWS servicios como Amazon S3 y DynamoDB. (AWS - Lambda, 2021)

2.2.5 DynamoDb

Amazon DynamoDB es una base de datos de clave-valor y documentos que ofrece rendimiento en milisegundos de un solo dígito a cualquier escala. Se trata de una base de datos completamente administrada, duradera, multiactiva y de varias regiones que cuenta con copia de seguridad, restauración y seguridad integradas, así como almacenamiento de caché en memoria para aplicaciones a escala de Internet. DynamoDB puede gestionar más de 10 billones de solicitudes por día y puede admitir picos de más de 20 millones de solicitudes por segundo. (AWS - Dynamodb, 2021)

2.2.6 CloudWatch

Amazon CloudWatch es un servicio de monitorización y observación creado para ingenieros de DevOps, desarrolladores, ingenieros de fiabilidad de sitio (SRE) y

administradores de TI. CloudWatch ofrece datos e información procesable para monitorizar sus aplicaciones, responder a cambios de rendimiento que afectan a todo el sistema, optimizar el uso de recursos y lograr una vista unificada del estado de las operaciones. CloudWatch recopila datos de monitorización y operaciones en formato de registros, métricas y eventos, lo cual ofrece una vista unificada de los recursos, las aplicaciones y los servicios de AWS que se ejecutan en servidores locales y de AWS. (AWS - CloudWatch, 2021)

2.2.7 Amazon Comprehend

Amazon Comprehend es un servicio de procesamiento de lenguaje natural (NLP) que utiliza el machine learning para descubrir información en datos no estructurados. En lugar de revisar los documentos, se simplifica el proceso y la información que no se ve es más fácil de entender.

El servicio puede identificar elementos fundamentales en los datos, incluidas referencias a idiomas, personas y lugares y los archivos de texto se pueden clasificar por temas relevantes. Puede detectar de forma automática y precisa la opinión de los clientes en su contenido en tiempo real. Esto acelera la toma de decisiones de manera más informada y en tiempo real para mejorar las experiencias de los clientes. Este servicio no solo localiza cualquier contenido que contenga información personal identificable, sino que también lo redacta y enmascara. Es un servicio completamente administrado por lo que no es necesaria configuración previa y además sin tener que entrenar modelos desde cero. (AWS - Comprehend, 2021)

2.3 Librería Tweepy



Figura 2.2. Tweeter logo

Es una librería con la cual podemos conectarnos con el API de Twitter de una manera sencilla utilizando el lenguaje Python. Esta librería proporciona diferentes funcionalidades que nos permiten:

- Autenticarnos contra el API de Twitter
- Realizar búsquedas de tweets
- Publicación de tweets
- Actualización del estado
- Contar likes
- Contar retweets
- Enviar mensajes directos
- Realizar conexiones en streaming para obtener tweets en tiempo real.

(Tweepy, 2021)

2.4 Serverless Framework



Figura 2.3. Serverless Framework

Serverless Framework nos provee de una capa de abstracción sobre los servicios de AWS en general y sobre AWS lambda en particular. Mediante su fichero yaml de configuración podremos describir el despliegue a realizar incluyendo las funciones lambda a crear, sus permisos de acceso y cómo van a interactuar con el resto de servicios del cloud de AWS como son API Gateway, S3, CloudFront, Route53, DynamoDB, etc.

Estas son sus principales características:

- Agnóstico del proveedor cloud empleado. Soportando tanto AWS, como Google y Azure.
- Orientado a componentes. Enfocado a construir integraciones con distintos elementos de la infraestructura y poderlas reutilizar de forma sencilla.
- Infraestructura como código. Define y despliega un conjunto de funciones y su interacción en una única operación atómica contextual al entorno o contexto seleccionado.

- Developer friendly. Centrado en la experiencia del desarrollador para maximizar su productividad.

(Serverless Framework, 2021)

Capítulo 3

Metodología y Desarrollo

3.1 Introducción

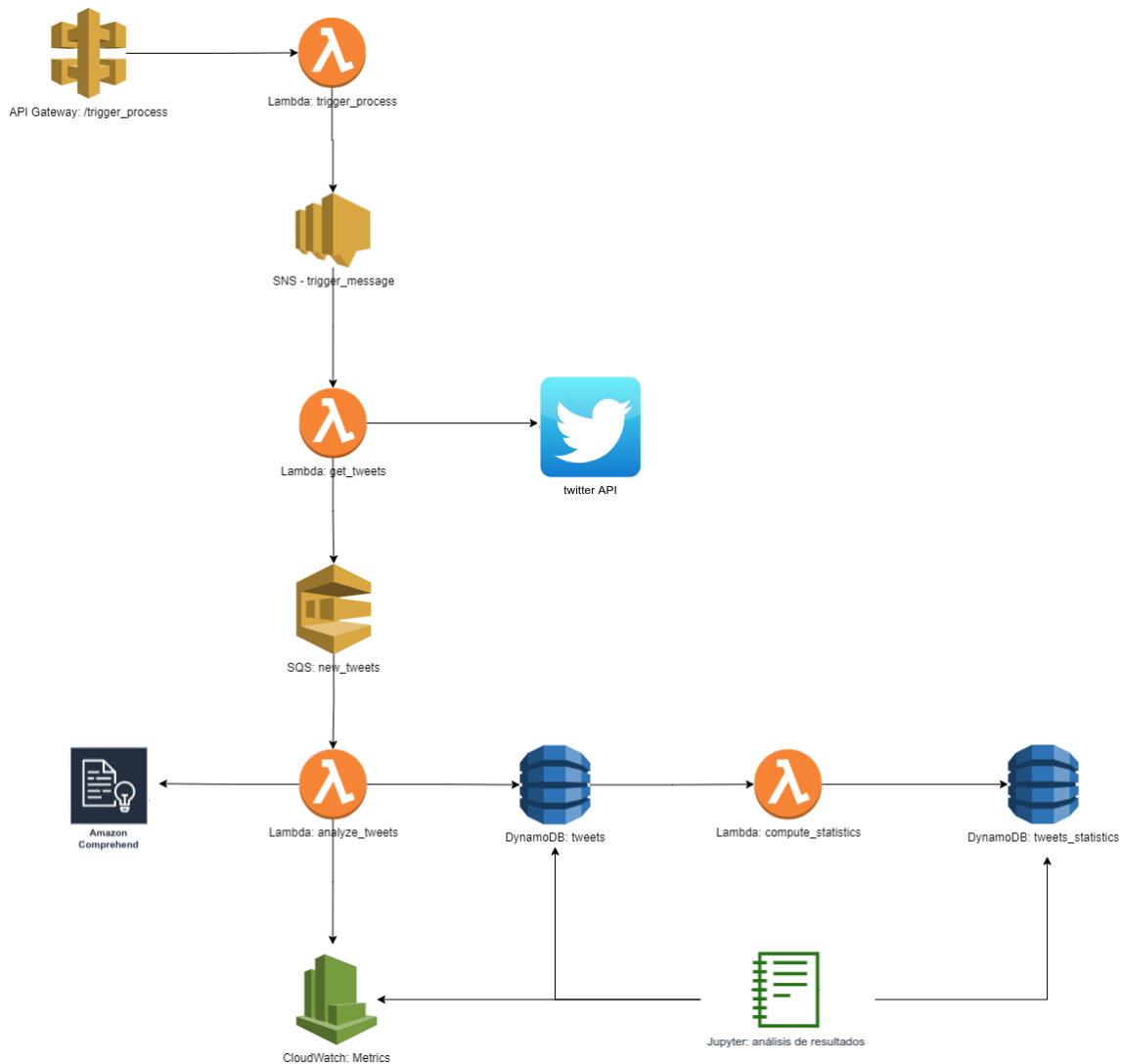


Figura 3.1. Arquitectura del sistema

En la figura de arriba podemos observar la arquitectura del sistema que se va a desarrollar. Para que el lector tenga una visión general del funcionamiento, a continuación se describe el flujo de trabajo del proceso completo:

- Un proceso automático hace una llamada a la API **/trigger process**
- Esta llamada dispara la ejecución de la función lambda **trigger_process**. Esta función registra un mensaje nuevo (**trigger_message**) en el servicio SNS.
- La función lambda **get_tweets** está suscrita al servicio SNS y será disparada cuando llegue un nuevo mensaje del tipo **"EXECUTE_PROCESS"**. Cuando esto sucede, la función lambda se conecta a la red social **Twitter** a través de su API y

hace una solicitud de 100 tweets. Conforme va recibiendo los tweets los envía a una cola de mensajes definida del servicio SQS (*new_tweets*).

- Por cada tweet registrado en la cola de mensajes SQS la función lambda **analyze_tweets** se ejecuta. Durante la ejecución el sistema hace uso del servicio Amazon Comprehend al que se le pasa el texto de cada uno de los tweets. Una vez este sistema analiza el texto del tweet devuelve la polaridad (sentimiento) de cada uno de ellos.
- Para evitar perder los datos generados, cada tweet junto con su polaridad se almacenan en una tabla en DynamoDB (*tweets*). Además, se genera un registro personalizado en CloudWatch para su posterior estudio.
- Con cada tweet que se añade a la tabla *tweets* se dispara la ejecución de la función lambda **compute_statistics** que simplemente hace un recuento por día de los tweets que han sido Positivos, Negativos, Neutros y Mixtos. Esta información se almacena finalmente en otra tabla de DynamoDB (**tweets_statistics**)

A continuación se detalla el desarrollo del sistema descrito primero haciendo uso directamente de los servicios que ofrece AWS a través de su consola de configuración. Posteriormente veremos cómo realizar este mismo trabajo haciendo uso de la tecnología Serverless Framework.

3.2 Desarrollo en AWS

A continuación se detalla el despliegue manual de cada uno de los servicios utilizados para la implementación de la arquitectura explicada más arriba:

Nota: Por simplificar la descripción de la implementación únicamente se comentará el código más representativo de cada una de las funciones lambda.

3.2.1 Lambda trigger_process

Para crear una función lambda el usuario selecciona el servicio y hace clic en el botón “Nueva función”. El sistema muestra la siguiente pantalla donde se define el nombre y el lenguaje que se va a usar:

Información básica

Nombre de la función

Escriba un nombre para describir el propósito de la

trigger_processs

Utilice exclusivamente letras, números, guiones o g

Tiempo de ejecución [Info](#)

Elija el lenguaje que desea utilizar para escribir la f

Python 3.8

Figura 3.2. Configuración básica de una función Lambda

Una vez creada, el usuario ya puede escribir el código que se ejecutará cada vez que la función es llamada. En concreto, esta función realiza las siguientes acciones:

- Crea un nuevo mensaje para enviarlo al servicio SNS:

```
sns_client.publish(TopicArn=topic_arn,  
                  Subject='EXECUTE_PROCESS',  
                  Message='Execute COVID sentiment analysis'  
                  )
```

Código 3.1. Publicación de un mensaje en el servicio SNS

Esta función lambda deberá ejecutarse con cada llamada que se haga a un punto de acceso definido. Para ello, en la pantalla de configuración, haciendo clic en el botón “*Agregar desencadenador*” podemos seleccionar el servicio API Gateway y crear el punto de acceso:



Figura 3.3. Desencadenador ejecución función Lambda

Una vez creado, podemos ver la URL del punto de acceso:

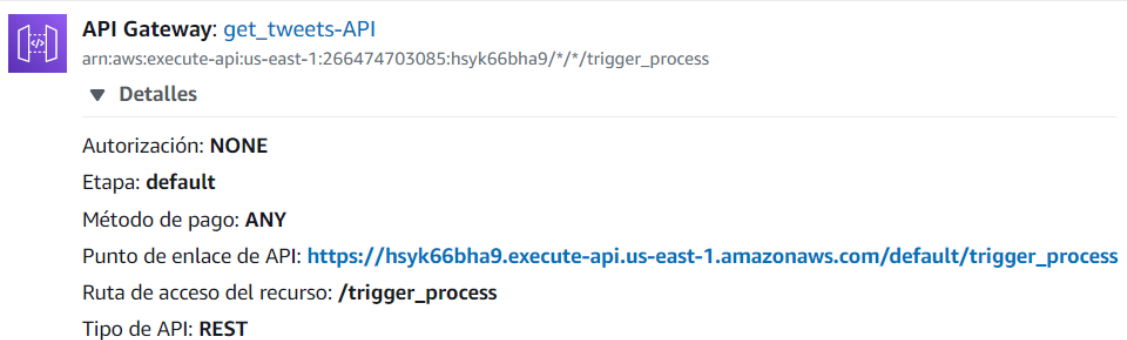


Figura 3.4. Información del punto de enlace de llamada a la API

Por último, para que la función lambda pueda enviar mensajes al servicio SNS es necesario asignarle permisos sobre este servicio. Por simplicidad se le ha otorgado full access:



Figura 3.5. Asociación permisos Full Access al servicio SNS

Nota: Para el resto de lambdas únicamente vamos a explicar el código implementado ya que la creación y asignación de permisos es exactamente igual.

3.2.2 Lambda get_tweets

El objetivo de esta función es conectarse a la red social Twitter y recopilar un conjunto de 100 tweets con cada ejecución.

Para asegurar la ejecución de esta función, la subscribimos al servicio de mensajería SNS, concretamente al topic definido en la función lambda anterior (EXECUTE_PROCESS):

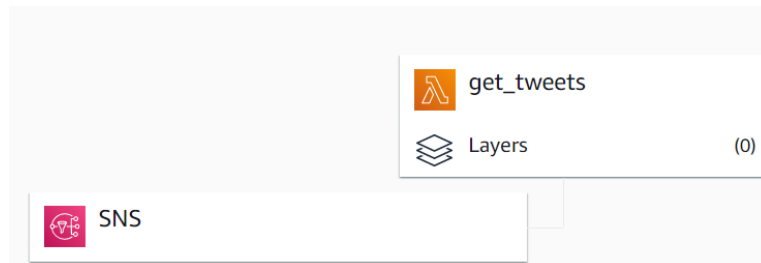


Figura 3.6. Desencadenador SNS de la ejecución de la función Lambda

Cada vez que un nuevo mensaje con dicho topic es publicado, la lambda se ejecuta realizando las siguientes acciones:

- Llamamos a la función search de la API tweepy para recuperar los 100 tweets más recientes relacionados con el COVID-19:

```
covid_tweets = api.search(key_words, lang='es', result_type='recent', count=100, include_entities=False)
```

Código 3.2. Llamada a la función search del API tweepy

La variable `key_words`, definida como variable de entorno, tiene el siguiente valor: *"CORONAVIRUS OR COVID OR COVID-19 OR SARS OR SARS-CoV OR VACUNA COVID"*. Por lo tanto todos los tweets que recuperamos estarán relacionados con la enfermedad.

- Cada uno de los tweets recibidos es enviado al servicio de colas de mensajes SQS para su posterior procesamiento.

```
response = sqs.send_message(
    QueueUrl=queue_url,
    MessageBody='{}'.format(tweet.id, remove_url(tweet_text))
)
```

Código 3.3. Envío de mensajes al servicio SQS

Al igual que en la anterior función, es necesario asignar los permisos necesarios a la lambda para que pueda acceder al servicio SQS. Una vez más, por simplicidad, hemos asignado el nivel de permisos full access:



Figura 3.7. Configuración de permisos de acceso al servicio SQS

3.2.3 Lambda analyze_tweets

Esta función es ejecutada cada vez que se recibe un mensaje a través del servicio SQS.

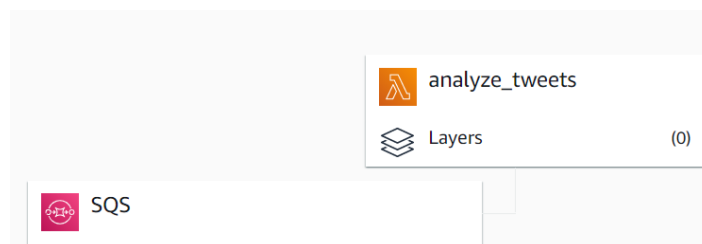


Figura 3.8. Desencadenador SQS de la ejecución de la función Lambda

Como ya hemos visto en la función anterior, cada mensaje contiene un tweet que deberá de ser analizado por esta función. Para ello hacemos también uso de uno de los servicios cognitivos de Amazon: Amazon Comprehend.

Una vez que tenemos el texto del tweet llamamos al servicio:

```
response = comprehend.detect_sentiment(Text=tweet_text, LanguageCode='es')
```

Código 3.4. Uso de la función `detect_sentiment` del API Comprehend

Posteriormente nos quedamos con la información devuelta a cerca del sentimiento:

```
# Sentimiento general del tweet
sentiment = response['Sentiment']
sent_pos = response['SentimentScore']['Positive']
sent_neg = response['SentimentScore']['Negative']

# Polaridad teniendo en cuenta la puntuación positiva y negativa
polarity = Decimal(sent_pos - sent_neg)
```

Código 3.5. Cálculo de los valores de sentimiento de un tweet concreto

Ahora la función almacena los datos en la tabla DynamoDb *tweets*:

```
table = dynamodb_client.Table('tweets')
table.put_item(
    Item={
        'tweet_id': tweet_id,
        'tweet_text': tweet_text,
        'sentiment': sentiment,
        'polarity': polarity,
```

```

        'tweet_date': datetime.today().strftime("%d/%m/%Y")
    }
)

```

Código 3.6. Añadiendo un registro a la tabla tweets

Por último, registra la información en el servicio CloudWatch para obtener gráficas que nos permitan analizar los resultados conforme vayamos obteniendo datos. En concreto, para cada día registramos la polaridad obtenida de cada uno de los tweets con el objetivo de tener un valor medio de la polaridad por día. Los valores oscilan en el rango [-1, 1] siendo -1 el valor más negativo y 1 el valor positivo:

```

namespace = 'cidaen_TFM_tomas'
dimension_sentiment = [
    {
        'Name': 'Sentiment Analysis',
        'Value': 'Polarity'
    }
]
cloudwatch.put_metric_data(
    MetricData = [
        {
            'MetricName': 'Co',
            'Dimensions': dimension_sentiment,
            'Unit': 'None',
            'Timestamp':  datetime(datetime.today().year,  datetime.today().month,
datetime.today().day),
            'Value': polarity,
        },
    ],
    Namespace=namespace

```

)

Código 3.7. Añadiendo un registro a CloudWatch

Al igual que en la anterior función, es necesario asignar los permisos necesarios a la lambda para que pueda acceder a los servicios de **Comprehend, DynamoDb, CloudWatch y SQS**. Una vez más, por simplicidad, hemos asignado el nivel de permisos full access:



Figura 3.9. Asociando permisos Full Access a la función Lambda

A modo de ejemplo, este sería el contenido de un registro en la tabla tweets:

tweet_date	tweet_id	polarity	sentiment	tweet_text
01/04/2021	137768231099736000	0,58	POSITIVE	VAMOS CON GANAS UnidosEnPrevencin Ganamos La Batalla Al Covid19

Tabla 3.1. Ejemplo registro tabla tweets en Dynamodb

3.2.4 Lambda compute_statistics

Con esta función lo que se busca es guardar un recuento por día del número de tweets Positivos, Negativos, Neutros y Mixtos. Estos resultados se cruzaran posteriormente con otra fuente de datos.

La ejecución se dispara cuando hay registros nuevos en la tabla tweets. En cada ejecución hace un recuento de los tweets de cada tipo:

```
if sentiment == 'POSITIVE':
    positive += 1
elif sentiment == 'NEGATIVE':
    negative += 1
elif sentiment == 'NEUTRAL':
    neutral += 1
else:
    mixed += 1
```

Código 3.8. Calculando valores acumulados por día

Una vez hecho el cálculo, crea un nuevo registro en la tabla tweets_statistics o actualiza el registro correspondiente si ya hubiera un registro con la misma fecha:

```
response = table.update_item(
    Key={
        'TweetDate': tweet_date,
    },
    UpdateExpression='ADD positive :val1, negative :val2, neutral :val3, mixed :val4',
    ExpressionAttributeValues={
        ':val1': positive,
```

```
        ':val2': negative,  
        ':val3': neutral,  
        ':val4': mixed  
    }  
)
```

Código 3.9. Actualizando valores en la tabla tweets_statistics

Al igual que en la anterior función, es necesario asignar los permisos necesarios a la lambda para que pueda acceder a servicio DynamoDb. Una vez más, por simplicidad, hemos asignado el nivel de permisos full access:



Figura 3.10. Asociación permisos Full Access sobre DynamoDB

A modo de ejemplo, este sería el contenido de un registro en la tabla tweets_statistics:

TweetDate	mixed	negative	neutral	positive
02/04/2021	11	22	56	10

Tabla 3.2. Ejemplo registro tabla tweets_statistics

3.3 Desarrollo con Serverless Framework

Para desarrollar una aplicación con Serverless Framework vamos a explicar los pasos a seguir:

3.3.1 Instalación

Para comenzar a trabajar con Serverless Framework, la primera tarea a realizar es instalar su dependencia globalmente a través de NPM:

```
npm install -g serverless
```

Código 3.10. Comando de instalación Serverles Framework

3.3.2 Creación de un proyecto Serverless

Una vez instalado, ya podemos crear el proyecto utilizando el comando serverless. El sistema generará una plantilla que nos sirve como base para definir nuestro proyecto. Como Serverless Framework nos permite definir funciones lambda implementadas en múltiples lenguajes, será necesario indicarle el lenguaje de trabajo (Python en este caso) y el nombre del proyecto (sentiment-analysis):

```
serverless create --template aws-python3 --name sentiment-analysis
```

Código 3.11. Comando de creación de un proyecto nuevo

Una vez ejecutado el comando anterior ya tenemos creado nuestro proyecto en la carpeta donde hayamos ejecutado dicho comando (otro parámetro a incluir sería la ruta donde queremos crear el proyecto si no queremos instalarlo en la carpeta desde donde ejecutamos el comando).

3.3.3 Archivo serverless.yml

Además de los ficheros de configuración, como por ejemplo el archivo package.json, se genera el archivo serverless.yml. Es en este fichero donde se define la arquitectura del sistema a desplegar.

A continuación se describe cada una de las partes de este fichero:

- En la captura de pantalla siguiente podemos ver cómo se define el nombre del servicio, la versión de Serverless Framework y el proveedor de servicios en la nube que vamos a utilizar:

```
# Nombre del servicio
service: sentiment-analysis

# Versión del framework
frameworkVersion: '2'

# Proveedor de servicios en la nube
provider:
  name: aws
  runtime: python3.8
  lambdaHashingVersion: 20201221
  stage: dev
  region: us-east-1
```

Figura 3.11. Configuración archivo yml I

- En la siguiente captura definimos las variables de entorno que serán transversalmente utilizadas por los servicios que vamos a implementar (en este caso únicamente la región donde vamos a desplegar la solución). Por otra parte también se definen los permisos que tendrán los roles asociados (por simplicidad y teniendo en cuenta que este no es un sistema que vayamos a utilizar en un ámbito profesional, se ha decidido otorgar full access para evitar problemas de acceso):


```
# Variables de entorno compartidas para todos los servicios
environment:
  REGION: us-east-1

# Configuración de permisos de acceso
iamRoleStatements:
  - Effect: "Allow"
    Action: "*"
    Resource: "*"

```

Figura 3.12. Configuración archivo yml II

- A continuación definimos las funciones lambda que se van a implementar. Nótese que todas las lambda se definen bajo la etiqueta functions:
 - o Función *trigger_process*: Para ello se crea una etiqueta con ese mismo nombre y debajo se definen las etiquetas:
 - **Handler** → Hace referencia al fichero handler_trigger.py donde se encuentra el código que deberá ejecutar la función lambda.
 - **Events** → Definimos el evento que desencadenará la ejecución de la función. En este caso hacemos referencia a la API *trigger_process* que será invocada con una llamada de tipo GET
 - **Environment** → Se define como variable de entorno la referencia al servicio SNS que va a utilizarse. Con la etiqueta Ref el sistema buscará en la sección de recursos la definición del servicio.

```
#Definición de funciones lambda
functions:
  trigger_process:
    handler: handler_trigger.trigger_process
    events:
      - httpApi:
          path: /trigger_process
          method: get
    environment:
      TOPIC_ARN: { Ref: MySNSTopic }

```

Figura 3.13. Configuración archivo yml III

- o Función *get_tweets*: Para ello se crea una etiqueta con ese mismo nombre y debajo se definen las etiquetas:

- **Handler** → Hace referencia al fichero `handler_tweets.py` donde se encuentra el código que deberá ejecutar la función lambda.
- **Timeout** → Extendemos el tiempo máximo de ejecución de la función a 30 segundos para asegurar que no se quedan tweets sin leer.
- **Events** → Definimos el evento que desencadenará la ejecución de la función. En este caso hacemos referencia al servicio SNS al que está suscrita la función.
- **Layers** → Para hacer uso de la API de `tweepy` es necesario empaquetarla y subirla a la lambda como layer.
- **Environment** → Se define como variable de entorno la referencia al servicio SQS que va a utilizarse para registrar los tweets recuperados. Por otra parte, se definen las credenciales de acceso para utilizar `tweepy`. Por último se define la variable de entorno `KEY_WORDS` que es una cadena de texto la cual vamos a usar para lanzar la búsqueda de tweets.

```
get_tweets:
  handler: handler_tweets.get_tweets
  timeout: 30
  events:
    - sns:
        arn: !Ref MySNSTopic
        topicName: trigger_message

  layers:
    - {Ref: TweepyLambdaLayer }

  environment:
    QUEUE_URL: { Ref: MySQS }
    USER_ID: H6UFCag0yfiYRjESSwdkMDQGC
    PASS_KEY: iqim2j0XvK8Q1WzgdMUTYcNUHfXDJ2fqcJvwkB56YEOVJOew9z
    ACCESS_TOKEN: 139041030-VCSNagPSbknNaoUxpoxFjHLUCDUBmDYlB9dXWxfI
    ACCESS_TOKEN_SECRET: gq4DYHBlHXJDQhc9pdBgOCG51L3UJnUwU0ZJpzcFMLPWI
    KEY_WORDS: '#CORONAVIRUS OR #COVID OR #COVID-19 OR #SARS OR #SARS-CoV'
```

Figura 3.14. Configuración archivo yml IV

- Función `analyze_tweets`: Para ello se crea una etiqueta con ese mismo nombre y debajo se definen las etiquetas:

- **Handler** → Hace referencia al fichero `handler_analysis.py` donde se encuentra el código que deberá ejecutar la función lambda.
- **Events** → Definimos el evento que desencadenará la ejecución de la función. En este caso hacemos referencia al servicio SQS al que está suscrita la función.
- **Environment** → Se define como variable de entorno la referencia a la tabla de DynamoDb que usaremos para almacenar los tweets analizados

```
analyze_tweets:
  handler: handler_analysis.analyze_tweets
  events:
    - sqs:
        arn:
          Fn::GetAtt:
            - MySQLS
            - Arn
  environment:
    TABLE_TWEETS: { Ref: DynamoTableTweets }
```

Figura 3.15. Configuración archivo yml V

- Función `compute_statistics`: Para ello se crea una etiqueta con ese mismo nombre y debajo se definen las etiquetas:
 - **Handler** → Hace referencia al fichero `handler_statistics.py` donde se encuentra el código que deberá ejecutar la función lambda.
 - **Events** → Definimos el evento que desencadenará la ejecución de la función. En este caso hacemos referencia al servicio DynamoDb de forma que cada vez que se introduce un registro en la tabla la función se ejecuta.
 - **Environment** → Se define como variable de entorno la referencia a la tabla de DynamoDb que usaremos para almacenar los resultados.

```

compute_statistics:
  handler: handler_statistics.compute_statistics
  events:
    - stream:
        type: dynamodb
        arn:
          Fn::GetAtt: [ DynamoTableTweets, StreamArn ]
  environment:
    TABLE_STATISTICS: { Ref: DynamoTableStatistics }

```

Figura 3.16. Configuración archivo yml VI

- A continuación bajo la etiqueta resources se definen todos los servicios que se usan en las funciones lambda descritas:
 - o Servicios **SNS y SQS**: Para ambos casos únicamente es necesario definir el nombre que le daremos a cada una de las instancias que creamos de estos servicios.

```

#Definición de recursos utilizados
resources:
  Resources:
    MySNSTopic:
      Type: AWS::SNS::Topic
      Properties:
        TopicName: trigger_message

    MySQS:
      Type: AWS::SQS::Queue
      Properties:
        QueueName: new_tweets

```

Figura 3.17. Configuración archivo yml VII

- o Servicio **DynamoDB**: Para nuestro proyecto se definen las dos siguientes tablas:
 - **Tweets** → Definimos el nombre de la tabla así como las clave primaria y la clave de ordenación. También se define bajo la etiqueta *StreamSpecification* el tipo de acción que hará que se lance la ejecución de la lambda asociada, en este caso cada vez que se crea un nuevo registro en la tabla

```

DynamoTableTweets:
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: tweets
    AttributeDefinitions:
      - AttributeName: tweet_id
        AttributeType: S
      - AttributeName: tweet_date
        AttributeType: S

    KeySchema:
      - AttributeName: tweet_date
        KeyType: HASH
      - AttributeName: tweet_id
        KeyType: RANGE

    ProvisionedThroughput:
      ReadCapacityUnits: 5
      WriteCapacityUnits: 5

    StreamSpecification:
      StreamViewType: NEW_IMAGE

```

Figura 3.18. Configuración archivo yml VIII

- **Tweets_statistics** → Definimos el nombre de la tabla así como las clave primaria y la clave de ordenación. En este caso no hace falta definir la etiqueta *StreamSpecification* puesto que esta tabla no va a lanzar ninguna ejecución posterior

```

DynamoTableStatistics:
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: tweets_statistics
    AttributeDefinitions:
      - AttributeName: TweetDate
        AttributeType: S

    KeySchema:
      - AttributeName: TweetDate
        KeyType: HASH

    ProvisionedThroughput:
      ReadCapacityUnits: 5
      WriteCapacityUnits: 5

```

Figura 3.19. Configuración archivo yml IX

- Por último definimos las dependencias que se deben incluir una vez publicamos la aplicación en AWS:
 - o **Etiqueta plugins** → De esta forma le decimos al sistema que las funciones lambda irán escritas en Python.
 - o **Etiqueta layers** → Aquí le decimos al sistema donde se encuentra la carpeta con todas las dependencias relativas a la API de Twitter

```
plugins:
  - serverless-python-requirements

# Definición de layer a usar donde se empaqueta la API de tweeter (tweepy)
layers:
  tweepy:
    path: ./layers
    description: Layer to use tweepy API
```

Figura 3.20. Configuración archivo yml X

3.3.4 Código de las funciones lambda

En referencia a los handlers definidos en cada una de las funciones vamos a mostrar, a modo de ejemplo, el código de una de las funciones lambda, concretamente el *handler_trigger.py*:

```
import boto3, json, os

sns_client = boto3.client('sns', region_name=os.environ['REGION'])
topic_arn = os.environ['TOPIC_ARN']

def trigger_process(event, context):
    sns_client.publish(TopicArn=topic_arn,
                      Subject='EXECUTE_PROCESS',
                      Message='Execute COVID sentiment analysis'
                      )

    body = {
        "message": "Function executed successfully!",
        "input": event
    }

    response = {
        "statusCode": 200,
        "body": json.dumps(body)
    }

    return response
```

Figura 3.21. Configuración archivo yml XI

La función *trigger_process* es la que ejecutará la lambda una vez se halla desplegado el servicio. Además, hay que destacar que las variables de entorno *REGION* o *TOPIC_ARN* son las definidas en el fichero *yml* de forma global y exclusivas de la función respectivamente.

Una vez que ya tenemos el fichero correctamente construido así como el código de las funciones lambda el siguiente paso es publicar el servicio.

3.3.5 Despliegue de la solución

Antes de nada deberemos tener correctamente configuradas nuestras credenciales para poder acceder a la cuenta AWS y tener permisos para publicar los servicios definidos. Para este ejemplo, actualizando el fichero *credentials* con la clave que se genera automáticamente al acceder a la cuenta AWS ya tendríamos acceso:

AWS CLI:

Copy and paste the following into ~/.aws/credentials

```
[default]
aws_access_key_id=ASIAZIKK56PUZE5DFFVY
aws_secret_access_key=0hW/h1y0aofCQL8akhqziK72e2rptbNLHTuFImB2
aws_session_token=IQ07b3jpZ2luX2VjEKD////////wEaCXVzLXdlc3QtMiJHMEUCIENCqjDMo1UviYBGkxi//4/vmUzhnaC/AZA0e2TFHb31AiEAr1115igvWm3QqZ4hEjeG3RdUJujWUazbr7h
W38qZcJcqsAITGRAAGgw2MzYzNDg0MjIwNTciDEENFVGhxamzDk07VCqNAoaTBNupqckxQ9tGNGNYA5KFHV/U+ZXXHpag7ND3krM9qGmk13Avg3UNPw5yMP0X/rBK29ItrA37XZkUiY8MBXZnPaEU+2iX
Yq7KPCzNvn1U0a4R9crZlMi1RP3F/MZ0uEwbATXz12p3kV4+/57GrLRkXJ9tqzstW0msF8y4cdPTeWzCxETg24Tii15nYg9UFZWxjRMfxvhaQbL7m5CFqBVSThI609HaQEi6+3DS5o1Esfi+3a8BGDSr
7UCAeGuQ+8ufGPC119u0wJyIx+sBYb1MA0qEu0ogdpKGD4tRYY9MnKsZQtY6FUwY8NwAIFDp55UcULIH0+R/DnUAbo5FbmEsb3A/1V+FJxwExdaMMOcloQG0p0BQd/asAphULNYEAHIKqDDAK3c4qnhDv
KgsmfMHLdJLA+L5Zzo0BZfrn7hexX1tAYi9yv+XGcZxuTnG95xruamq1zHg9FiVKCF0nBjs/Tjo2GKQBgYmz50G5qdf3cXDznQ/k+Cru5dA0dXfCX5DM+4HTteF6bPF6z0MwDIIo73RTolL800Y3A5Jj
j6bb/fXWUpZw7dJjWbmn7tz9tg==
```

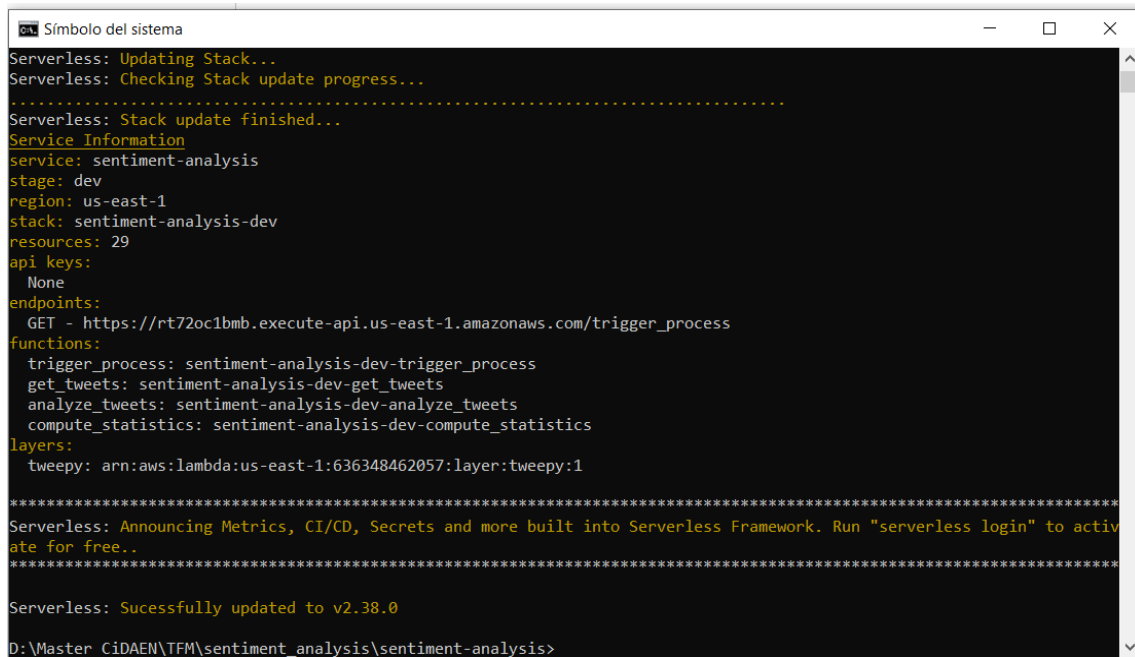
Figura 3.22. Credenciales AWS

Una vez tenemos las credenciales configuradas correctamente, simplemente hay que ejecutar el siguiente comando dentro de la carpeta donde tenemos el proyecto:

serverless deploy

Código 3.12. Comando de despliegue del servicio en AWS

Una vez terminado el despliegue el sistema muestra un mensaje como el siguiente:



```
Símbolo del sistema
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.....
Serverless: Stack update finished...
Service Information
service: sentiment-analysis
stage: dev
region: us-east-1
stack: sentiment-analysis-dev
resources: 29
api keys:
None
endpoints:
GET - https://rt72oc1bmb.execute-api.us-east-1.amazonaws.com/trigger_process
functions:
trigger_process: sentiment-analysis-dev-trigger_process
get_tweets: sentiment-analysis-dev-get_tweets
analyze_tweets: sentiment-analysis-dev-analyze_tweets
compute_statistics: sentiment-analysis-dev-compute_statistics
layers:
tweepy: arn:aws:lambda:us-east-1:636348462057:layer:tweepy:1
*****
Serverless: Announcing Metrics, CI/CD, Secrets and more built into Serverless Framework. Run "serverless login" to activate for free..
*****
Serverless: Successfully updated to v2.38.0
D:\Master CIDAEN\TFM\sentiment_analysis\sentiment-analysis>
```

Figura 3.23. Informe de publicación del servicio en AWS

En la captura podemos ver toda la información de la publicación del servicio. Por ejemplo:

- Nombre del servicio.
- Etapa de publicación (podemos tener diferentes publicaciones separando entornos de desarrollo (dev), pruebas y producción).
- Región donde está publicado el servicio.
- Puntos de entrada → En nuestro caso, la API a través de la cual disparamos el proceso completo.
- Conjunto de funciones lambda creadas.
- Layers creadas para la correcta ejecución de las funciones.

3.3.6 Eliminación del servicio

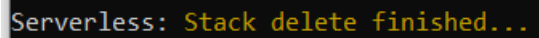
Serverless proporciona una función que nos permite eliminar todo rastro de los servicios que hemos publicado dentro de un mismo proyecto. De esta forma el usuario se ahorra el tiempo que conllevaría ir eliminando uno a uno todos los servicios utilizados.

Simplemente escribiendo el siguiente comando eliminaríamos la aplicación completamente:

serverless remove

Código 3.13. Comando de eliminación del servicio en AWS

Una vez finalizado el proceso el sistema muestra este mensaje:



```
Serverless: Stack delete finished...
```

Figura 3.24. Confirmación de la eliminación de los servicios en AWS

3.4 Conclusiones

Una vez desarrollado el sistema haciendo uso de las dos formas descritas tengo que decir que, sobre todo para un entorno profesional, me decantaría por utilizar Serverless Framework ya que acelera enormemente las actividades de desarrollo y despliegue de estos sistemas.

Además, con un simple comando podemos replicar la misma arquitectura y funcionalidad tanto en diferentes cuentas AWS como en la misma teniendo diferentes entornos (testeo, validación y producción) totalmente aislados.

Capítulo 4

Experimentos y Resultados

4.1 Introducción

Una vez desarrollado y desplegado el sistema descrito en el capítulo tres se inició la ejecución de la aplicación el día 15/03/2021. A partir de ese día se ha llevado una ejecución diaria.

Con los datos obtenidos de la red social Twitter se han ido registrando datos en el servicio **CloudWatch** de AWS para tener una representación gráfica de la evolución de la polaridad durante todo este tiempo.

Por otra parte, se han obtenido datos actualizados de un fichero csv cuya fuente es el Centro Nacional de Epidemiología que cruzaremos con los datos obtenidos de Twitter. Para la exploración de estos datos y comparación de los mismos se ha desarrollado una libreta Jupyter cuyo contenido vamos a ver de una forma resumida a lo largo de este capítulo.

4.2 Servicio CloudWatch

A continuación vamos a ver un conjunto de gráficas que representan la polaridad obtenida durante la ejecución de la aplicación.

El valor de polaridad para cada uno de los días oscila entre los valores **[-1, 1]** donde -1 representa el sentimiento más negativo y 1 representa el sentimiento más positivo.

Primero vamos a ver la representación gráfica de los valores máximos por día, es decir, los valores con sentimiento más positivo:

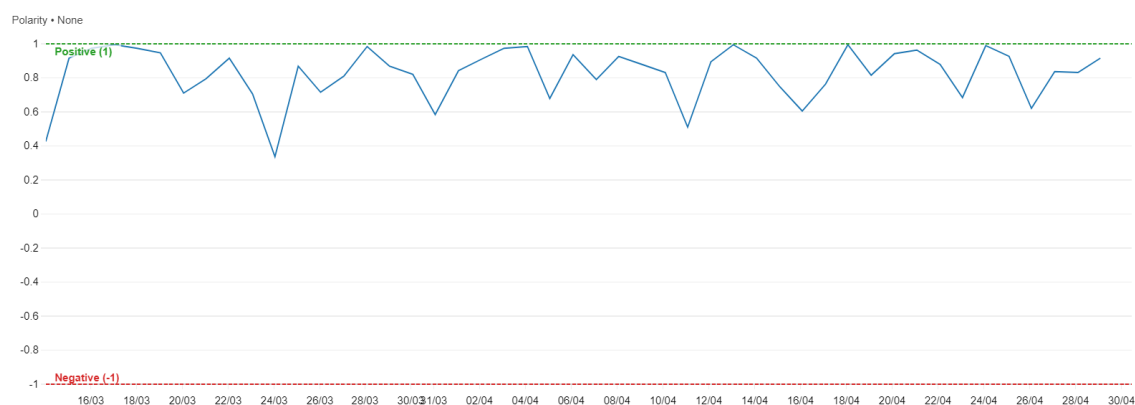


Figura 4.1. Gráfica polaridad máximos por día

Ahora la representación gráfica de los valores mínimos por día, es decir, los valores con sentimiento negativo:

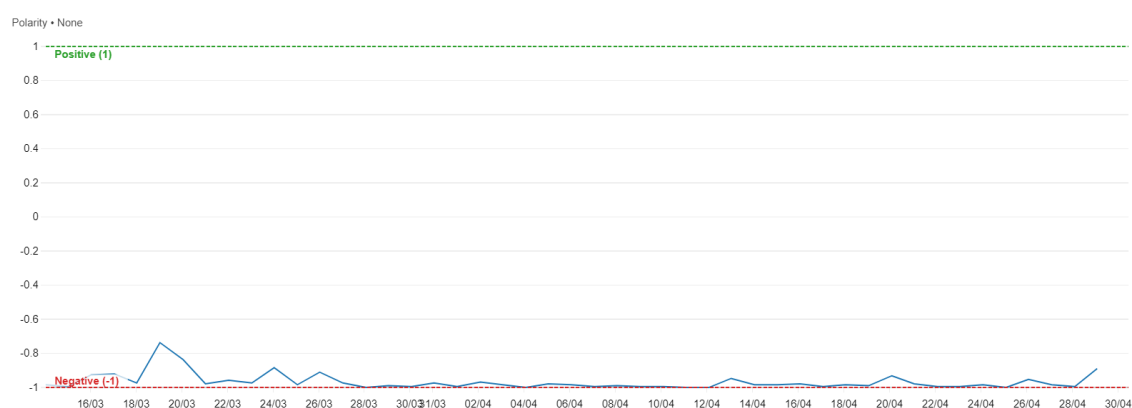


Figura 4.2. Gráfica polaridad mínimos por día

Finalmente obtenemos una representación gráfica de los valores medios por día:

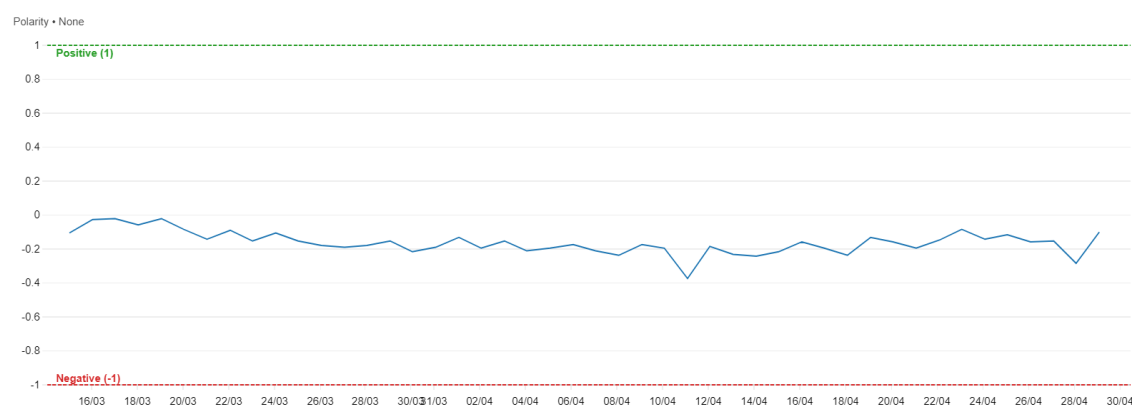


Figura 4.3. Gráfica polaridad valores medios por día

Como podemos observar, los valores medios obtenidos se mantienen por debajo de 0. Podemos decir por lo tanto que el sentimiento durante el período estudiado se mantiene en un valor negativo.

4.3 Libreta Jupyter

A modo de ejemplo, un pequeño extracto del conjunto de datos del Centro Nacional de Epidemiología:

	provincia_iso	sexo	grupo_edad	fecha	num_casos	num_hosp	num_uci	num_def
0	A	H	0-9	2020-01-01	0	0	0	0
1	A	H	10-19	2020-01-01	0	0	0	0
2	A	H	20-29	2020-01-01	0	0	0	0
3	A	H	30-39	2020-01-01	0	0	0	0
4	A	H	40-49	2020-01-01	0	0	0	0

Figura 4.4. Muestra del set de datos

Los datos vienen separados por Fecha / Provincia / Sexo / Grupo de edad. Las columnas que vamos a estudiar son:

- **num_casos:** Número de casos
- **num_hosp:** Número de hospitalizados
- **num_uci:** Número de ingresados UCI
- **num_def:** Número de defunciones

Para nuestro análisis agruparemos únicamente por fecha para obtener los totales de las columnas detalladas arriba por día.

Una vez hemos filtrado el conjunto de datos por los datos obtenidos a partir del día 15/03/2021, agrupamos y sumamos totales por fecha:

	num_casos	num_hosp	num_uci	num_def
fecha				
2021-03-15	5414	442	52	94
2021-03-16	5971	424	60	101
2021-03-17	5697	460	53	86
2021-03-18	6281	427	53	65
2021-03-19	4570	389	48	57

Figura 4.5. Valores totales por día

Si imprimimos una gráfica estos son los resultados:

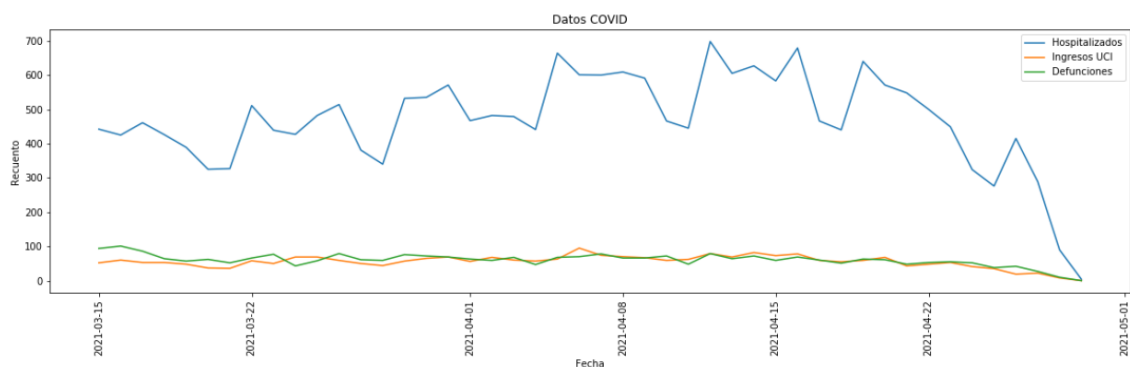


Figura 4.6. Hospitalizaciones, ingresos UCI y defunciones

El siguiente paso sería realizar un scan de la tabla *tweets* que tenemos en AWS para obtener un conjunto de datos como el que sigue:

	polarity	tweet_id	sentiment	tweet_date	tweet_text
0	-0.577922	137768230066259968	NEGATIVE	2021-04-01	ltimos 130 dasCasos covid sin sancin ni comuni...
1	-0.345516	137768230082623078	NEUTRAL	2021-04-01	Colombia reporta 8646 nuevos casos de Covid19 ...
2	0.024719	137768230236552806	NEUTRAL	2021-04-01	El 591 de la poblacin espaola ha recibido la v...
3	-0.015667	137768230242415411	NEUTRAL	2021-04-01	LTIMAHORA Cardenal Baltazar Porras anunci este...
4	-0.028315	137768230247034061	NEUTRAL	2021-04-01	Magallanes hoy reporta 28 casos nuevos de Covi...
...
3853	0.128540	137695757826617344	NEUTRAL	2021-03-30	El ozono puede ser una solucin para desinfecta...
3854	-0.420907	137695757839198208	MIXED	2021-03-30	Democrticamente esperando la vacuna contra el ...
3855	-0.043927	137695757923509863	NEUTRAL	2021-03-30	Cuidmonos La Alcaldia de Bruzual a travs del re...
3856	-0.842154	137695757939029608	NEGATIVE	2021-03-30	Finalmente soy Covid Tengo ambas dosis de Coro...
3857	-0.552001	137695757960415232	NEGATIVE	2021-03-30	NotiMippCI Vicepresidenta Rodriguez El mundo vi...

Figura 4.7. Ejemplo contenido tabla tweets en Dynamodb

Agrupando por fecha y calculando la media del campo **polarity**, podemos hacer un cruce de estos datos con los obtenidos en la gráfica anterior:

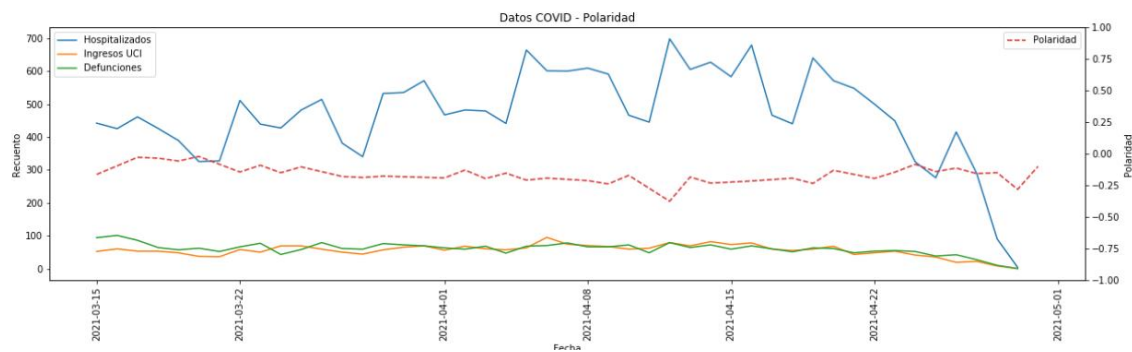


Figura 4.8. Hospitalizaciones, ingresos UCI y defunciones VS Polaridad

En esta gráfica vemos que no se puede establecer a priori la correlación lógica esperada: A una mejora de la situación (disminución de contagios, hospitalizaciones y muertes), una progresión de la polaridad con tendencia a acercarse al valor 1. Esto puede deberse a la cantidad demasiado elevada de tweets clasificados como neutros o mixtos.

Por último, usando la tabla *tweets_statistics* vamos a intentar ver si hay algún tipo de relación entre el ratio de tweets positivos y negativos con respecto al total y compararlo con la polaridad. Para ello, hacemos un scan de la tabla para obtener los datos:

	neutral	mixed	TweetDate	negative	positive
0	55	17	01/04/2021	22	3
1	76	13	20/03/2021	7	4
2	98	37	05/04/2021	58	6
3	54	17	03/04/2021	26	2
4	71	10	25/03/2021	18	0

Figura 4.9. Muestra contenido tabla *tweets_statistics* en Dynamodb

Ahora creamos dos nuevas columnas con el ratio, agrupamos por fecha, calculamos totales y cruzamos con los datos de polaridad:

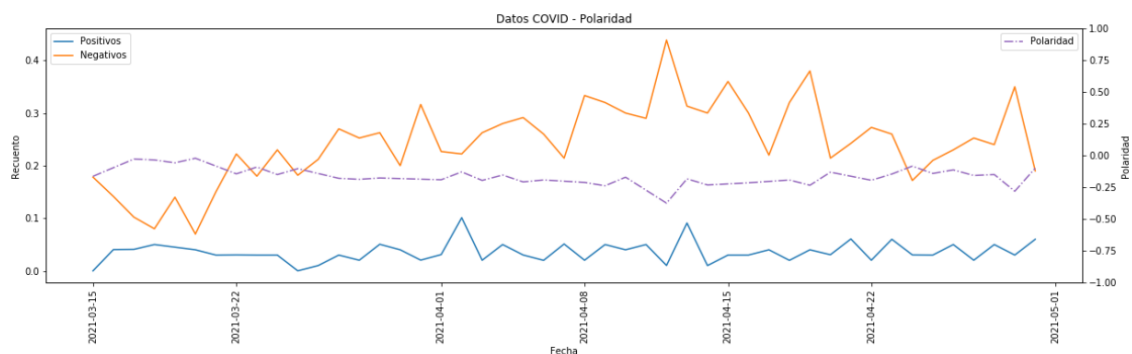


Figura 4.10. Tweets positivos - negativos VS Polaridad obtenida

En esta última gráfica vemos cómo la polaridad está por debajo de 0 en todo momento, lo cual es lógico dado el número de tweets negativos que siempre ha superado ampliamente el número de tweets positivos.

Nota: Para ver el contenido de la libreta completa se puede descargar desde aquí: [Anexo](#)

[1](#)

Capítulo 5

Conclusiones y Trabajo Futuro

5.1 Conclusiones

Una vez realizado este Trabajo Fin de Máster, podemos afirmar que se han cumplido todos los objetivos fijados inicialmente:

- Se ha desarrollado cada una de las partes de la arquitectura presentada haciendo uso de la consola de AWS.
- Se ha desarrollado la misma arquitectura haciendo uso de Serverless Framework permitiéndonos ver las bondades de esta tecnología.
- Se ha realizado una extracción de datos de una red social como es Twitter. Serían necesarias mejoras adicionales para obtener conjuntos de datos que nos permitieran obtener datos más representativos.
- Se ha realizado un almacenamiento y transformación de los datos extraídos.
- Se ha realizado un análisis de sentimiento haciendo uso del servicio Amazon Comprehend.
- Se ha creado una libreta Jupyter en la que se comparan los datos de la red social con los datos obtenidos del Centro Nacional de Epidemiología.

Gracias a la consecución de estos objetivos, el alumno se ha iniciado en tecnologías como Serverless Framework la cual desconocía por completo. Además, el alumno ha profundizado en el uso de servicios en la nube descubriendo algunos de ellos como SNS, SQS y Amazon Comprehend.

5.2 Trabajo futuro

Como trabajo futuro se podría realizar:

- Estudio de técnicas más avanzadas del procesamiento del lenguaje para la obtención de datos más fiables en el estudio que se ha realizado.
- Mejora en la descarga de tweets utilizando un stream de datos en tiempo real y utilizando servicios AWS como por ejemplo Kinesis.
- Realización de una aplicación web en la que mostrar los resultados obtenidos ofreciendo al usuario diferentes cuadros de mandos con la opción de poder configurar filtros, generación de reportes, etc...

Bibliografía

Amazon Web Services. (Abril de 2021). Obtenido de AWS: <https://aws.amazon.com/es/>

AWS - API Gateway. (Abril de 2021). Obtenido de <https://aws.amazon.com/es/api-gateway/>

AWS - CloudWatch. (Abril de 2021). Obtenido de <https://aws.amazon.com/es/cloudwatch/>

AWS - Comprehend. (Abril de 2021). Obtenido de <https://aws.amazon.com/es/comprehend>

AWS - Dynamodb. (Abril de 2021). Obtenido de <https://aws.amazon.com/es/dynamodb/>

AWS - Lambda. (Abril de 2021). Obtenido de <https://aws.amazon.com/es/lambda/>

AWS - SNS. (Abril de 2021). Obtenido de <https://aws.amazon.com/es/sns/>

AWS - SQS. (Abril de 2021). Obtenido de <https://aws.amazon.com/es/sqs/>

Serverless Framework. (Abril de 2021). Obtenido de <https://www.serverless.com/framework/docs/>

Tweepy. (Abril de 2021). Obtenido de <https://www.tweepy.org/>

Anexo I. Recursos del proyecto

I.1 Recursos

En la siguiente URL se encuentra el repositorio en GitHub donde se puede encontrar:

- Libreta Jupyter con los resultados obtenidos
- Proyecto Serverless Framework listo para ser desplegado en cualquier cuenta de AWS

https://github.com/tomaspc83/MasterCiDAEN_TFM