Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO TUTELADO - EII
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DE COMPUTADORES

AWS Lambda: Estudo xeral + Caso práctico

Estudante: Mauro A. de los Santos Nodar

A Coruña, xaneiro de 2021.

Índice Xeral

Índice de Figuras

Capítulo 1

Introdución

Neste traballo tutelado tentarase facer una revisión xeral sobre o que é, as partes e características que presenta e as posibilidades que ofrece a ferramenta de *Amazon Cloud Computing* de **AWS Lambda**. Por outro lado, despois de facer esta parte teórica, buscarase exemplificar o explicado levando a cabo un breve proxecto práctico que faga uso desta ferramenta creando un ***bot* de *Twitter* en *Python*** que use datos de ***Spotify***.

Capítulo 2

Análise Teórica

2.1 Introdución a AWS Lambda

Imos para empezar, ver unha breve introdución do que é AWS Lambda e dos principais elementos que o caracterizan:

AWS Lambda é un **servizo informático** que permite executar código en base a eventos sen ningún tipo de aprovisionamento ou administración de servidores, pagando só polo tempo de cómputo consumido. Pode executar practicamente calquera tipo de código, o cal só haberá que cargalo e o propio Lambda será o encargado de executalo en base a, como ben comentamos, múltiples tipos de eventos ou desencadeadores (dende *HTTP requests* ata cambios en datos de *Amazon S3 buckets* ou como se verá máis adiante neste traballo, ao estilo ***Cron*** grazas a *EventBridges*), todo isto á vez que administra automáticamente todos os recursos informáticos necesarios. [1]

É dicir, Lambda executará o código tan só cando sexa necesario, é dicir, cando os nosos eventos configurados o requiran, e faráo ademais nunha infraestrutura informática de alta dispoñibilidade onde se encargará de todo o referente á administración dos recursos informáticos: como son o mantemento do servidor ou do OS, o aprovisionamento de capacidade e o escalado automático, a implementación de seguridade e a monitorización do código, entre moitas otras cousas. O único que terá que facer o cliente será proporcionar o código e pagar polo tempo de computación que consume. [2]

2.2 Cando usar Lambda

Cada día, o servizo informático de Lambda é máis e máis utilizado xa que cada vez hai máis usuarios que buscan escalabilidade, rendemento e eficiencia de custo sen ter que lidiar ca xestión da infraestrutura. Lambda será unha plataforma para múltiples e diversas aplicacións que fagan uso de diferentes linguaxes e *runtimes* soportados polo propio servizo, polo

que cando un usuario necesite ou queira poñer todo o seu esforzo unicamente na tarefa de *code-making*, é dicir, de programar código, e non queira lidiar con todo o mantemento e aprovisonamento de servidores, escalabilidade, concorrencia, etcétera, debería considerar AWS Lambda como a súa opción principal. Polo que vemos aquí como de forma xenérica, Lambda pode adaptarse a milleiros de casos de uso, pero imos aínda así, mencionar unha serie deles de forma representativa onde podemos sacarlle o máximo partido a esta ferramenta: [2]

- **Servizos back-end personalizados**, con código propio ou orientados a accións ou eventos que non teñen que estar correndo indefinidamente, como veremos máis adiante no exemplo deste propio traballo.
- **Creación de APIs** xunto *Amazon API Gateway*.
- **Creación de tarefas de tipo Cron** con *CloudWatch* para a automatización de procesos, axudado todo isto pola non-limitación de recursos de Lambda que escalará automaticamente o que sexa necesario.

2.3 Características e vantaxes

Vemos pois que os casos donde o uso de AWS Lambda é acertado son infinitos debido á gran cantidade de posibilidades e vantaxes que nos ofrece. Polo que imos ver en detalle as partes e características principais de AWS Lambda: [1] [3]

- Non hai servidores que **administrar nin aprovisionar**. Como ben comentamos xa, AWS Lambda **executa automáticamente** o código.
- **Escalado continuo**. AWS Lambda escala automaticamente a aplicación mediante a execución de código en resposta a cada desencadeador ou *trigger* que definamos. Todo este código é executado en paralelo e procesado por cada desencadeador de xeito separado (verémolo máis adiante na imaxe 2.1).
- Un das súas chaves será a **execución baseada en eventos**, é dicir, cando os eventos configurados ocorran, Lambda executará a función ou funcións configuradas e administrará os recursos de cómputo segundo sexa necesario para atender todas as solicitudes entrantes en tempo real, brindando un **rendemento consistente**.
- Pódese **usar en conxunto con outros servizos AWS** como os que vimos na asignatura para ampliar funcionalidades, ou ben de maneira individual e personalizada como comentamos previamente con eventos e código propio.

- Contará con **tolerancia a erros** integrada, así como con funcionalidades referentes á **seguridade** e á **monitorización** do noso código, as cales veremos tamén ao final deste capítulo.
- En canto ao apartado de *Networking*, destacar que de forma predeterminada Lambda executará as súas funcións nunha **nube privada virtual (VPC)**, como ben vimos nas prácticas de AWS da asignatura. Esta será unha nube interna con conectividade tanto aos servizos de AWS como a Internet.
- Por último, destacar algo do **prezo ou custo** para o cliente deste servizo, mencionando que se aplicarán cargos aos clientes por cada 100 ms de execución do código e polo número de veces que este sexa activado.

2.4 Especificacións

Haberá que mencionar rapidamente as **especificacións técnicas** de AWS Lambda, o cal soportará os seguintes entornos de execución: [4] [5]

Linguaxe	Versións
Node.js	10 - 12
Java	8 - 11
Python	3.6 - 3.7 - 3.8 - 2.7
Go	1.X
Ruby	2.5 - 2.7
.NET Core	2.1 - 3.1

E o cal executará cada función nun contedor con *Amazon Linux AMI a 64-bit.* Onde o entorno de execución contará con: [4][5]

Característica	Valor
Memoria	128MB - 3008MB, en incrementos de 64 MB
Espacio en disco efímero	512MB
Duración máxima de ejecución	900 segundos
Tamaño del paquete comprimido/descomprimido	50/250 MB

2.5 Funcionamento en detalle

Imos agora despois de ver as características principais, entrar un pouco máis en detalle no funcionamento das funcións Lambda e da organización dos entornos de execución.

Cada función correrá nun ou máis **entornos de execución** dedicados, usados tan só durante o tempo de vida da función e despois destruídos. Cada entorno ocuparáse dunha única execución concorrente, pero poderá ser reutilizado para a execución múltiple da mesma función de forma secuencial.

Estes entornos correrán en ***microVMs***, as cales son máquinas virtuais dedicadas de pequeno tamaño. Cada unha destas está asociada a unha conta AWS, pero pode ser reutilizada por varios entornos de execución desa mesma conta.

Todas estas son empaquetadas e gobernadas mediante unha plataforma *hardware* chamada **Lambda Workers**, que non será máis ca unha **instancia EC2** como as usadas nas prácticas da asignatura.

En canto á **compartición cruzada**, dicir que os entornos de execución nunca compartirán funcións e *microVMs* desta forma, e que as *microVMs* non compartirán AWS accounts.

Todo isto podemos velo de forma gráfica e máis clara na seguinte imaxe: [2]

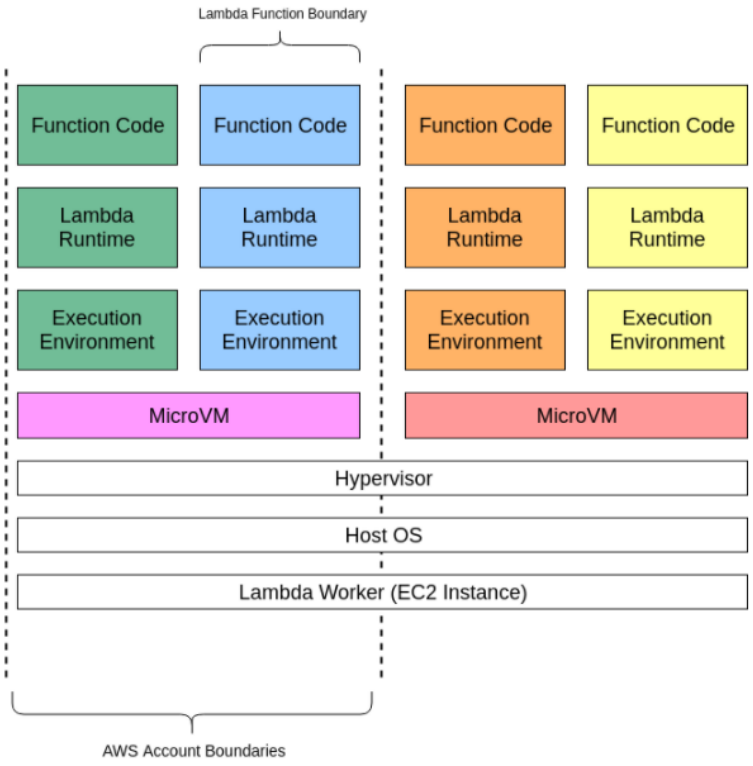

Figura 2.1: Como funciona a execución de funcións en AWS Lambda

En canto ao código que se executa en AWS Lambda, como xa vimos chámaselle ***Lambda Function***. Despois de crear unha destas, sempre estará lista para executarse en canto se active debido a un evento que nós elixamos previamente. Cada función inclúe código e certa información de configuración asociada, como son o nome da función, os requisitos en materia de recursos pero sobretodo, parámetros que veremos no caso práctico no seguinte capítulo como son as **variables de entorno**, configuradas directamente sobre a consola AWS e as **librerías** utilizadas pola nosa función *Lambda*, especificadas mediante a creación dunha capa ou **Layer** que conste de todas elas e esté asociada á función que as necesite.

Destacar tamén neste capítulo que as funcións de *Lambda* **non teñen estado**, polo que *Lambda* pode lanzar tantas copias da función como resulten necesarias para escalar adecuadamente ao índice de eventos entrantes. Esto tradúcese tamén en que cada vez que a función *Lambda* é disparada por un un evento, invócase nun entorno completamente novo. Aínda así, como comentamos anteriormente, a función invócase só unha vez por contedor, entón, cando se invoca por primeira vez, todo o código na nosa función é executado para despois ser invocado, pero nas seguintes execucións, se o contedor aínda está dispoñible, invócase directamente a función e non todo o código de novo.

Por último, facer de novo un pouco de énfase no **autoescalado** e no comportamento dos **contedores**, os cales son administrados completamente por AWS. Actívanse cando se produce un evento e apagánse se non se están a utilizar. Se se realizan solicitudes adicionais mentras se está servindo o evento orixinal, como ben dixemos, iníciase un novo contedor para atender a solicitude, esto significa que si estamos experimentando un pico de uso, AWS simplemente creará varias instancias do contedor coa nosa función para atender esas solicitudes. Neste parágrafo podemos ver explicados os conceptos de **escalado automático**, ***server-less*** e **concorrencia** dos cales falamos ao longo de toda a memoria. É certo que, sobre todo isto, Amazon fará algunhas optimzacións propias, como por exemplo manter o contedor durante uns minutos (de 5 a 15) para que poida responder ás solicitudes inmediatamente posteriores sen un arranque dende 0 de novo.

2.6 Conceptos complementarios

2.6.1 Monitorización

Lambda monitorea automaticamente as nosas funcións e informa sobre as métricas a través de **Amazon CloudWatch**. Para axudar a monitorizar o código mentras se executa, *Lambda* controla tamén automaticamente o número de solicitudes, a duración da invocación de cada solicitude e o número de solicitudes que xeran un erro. *Lambda* publicará tamén as métricas de *CloudWatch* asociadas, por exemplo para configurar alarmas personalizadas como ben vimos na última práctica da asignatura. [1]

A propia consola *Lambda* proporcionará un **panel de monitorización** integrado para cada unha das súas funcións e aplicacións, o cal veremos exemplificado na nosa propia función ao final desta memoria (3.21).

Á parte de *CloudWatch*, haberá tamén mais opcións para a monitorización como poden ser, entre outras: [2]

- **AWS X-Ray**: Ofrecerá unha vista de como viaxan as peticións ao longo da nosa aplicación e un mapa do rendemento de todos os compoñentes da mesma.
- **AWS Config**: *Trackeará* todo tipo de cambio no proxecto: configuración das funcións, entornos de execución, código, axustes de concorrencia, variables de entorno, etcétera.

2.6.2 Seguridade

A seguridade en AWS *Lambda* será un curioso concepto que xogará ca idea **reponsabilidade compartida** entre AWS e o cliente. Este modelo diferencia a seguridade en: seguridade **da** nube e seguridade **na** nube: [1]

- **Seguridad da nube**: AWS é o responsable de protexer a infraestrutura que executa os servizos de AWS na nube mediante auditores externos que proban e verifican periodicamente a eficacia da seguridade.
- **Seguridad na nube**: É responsabilidade do cliente a confidencialidade dos datos, os requisitos da empresa, a lexislación, etcétera. Tamén é responsable da seguridade do seu propio código, e do acceso á sua función, como é obvio.

Aínda así podemos ver graficamente e de forma máis detallada este modelo de seguridade compartida que propón AWS: [2]

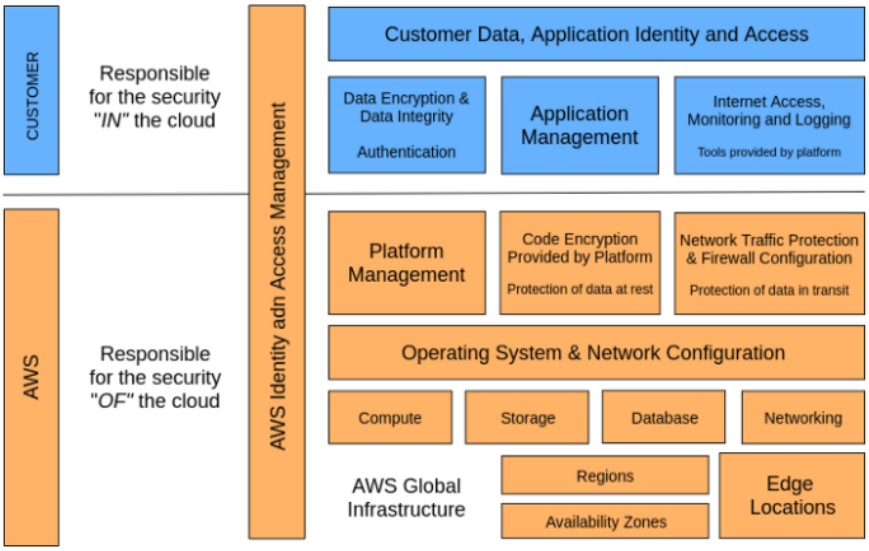

Figura 2.2: O modelo de seguridade compartida en AWS Lambda

2.7 Resumo

Polo que resumindo graficamente o seu funcionamento podería ser o seguinte:[1]

- Sube o teu **código funcional** = *Function* (xunto coas súas **librerías e dependencias** = *Layer*).
- Establece **eventos** á túa medida que executen ese código (desencadeadores = *Triggers*).
- **Relaciona os tres elementos**: función, capa e desencadeador/es e deixa que AWS se encargue da posterior execución e administración.
- Paga só polo tempo e recursos que foron utilizados ao final do mes.

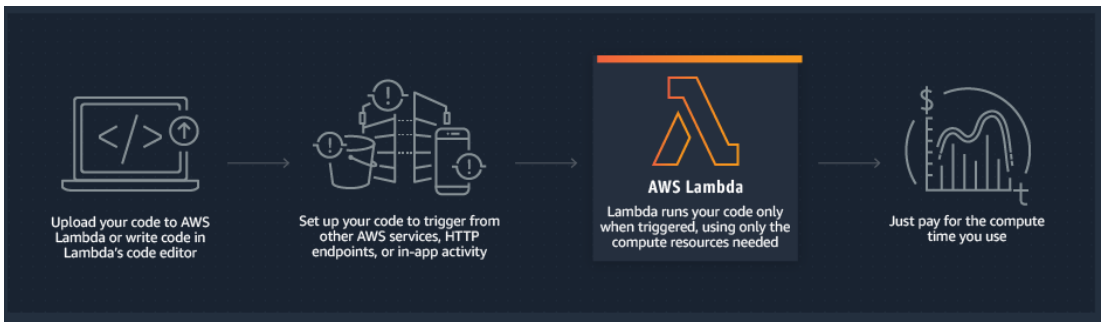

Figura 2.3: Pasos para o uso xenérico de AWS Lambda

Polo que chegados a este punto vimos xa unha aproximación bastante completa sobre o que é, como funciona e as diferentes partes e características que forman ***AWS Lambda***, tanto de forma xenérica, nas primeras seccións como de forma moito máis detallada nas últimas. Polo que imos ver agora un exemplo concreto de todo o exposto ca seguinte parte da memoria, onde levaremos a cabo un proxecto que use este servizo e mostraremos paso a paso como despregalo e poñelo en marcha nun entorno real de produción, mostrando así o código completo, capturas de pantalla traballando coa consola de *Lambda* e os resultados das execucións na mesma. Polo que, imos pois, á parte práctica do proxecto.

Capítulo 3

Caso Práctico

3.1 Introdución ao proxecto

O proxecto a desenvolver consistirá no despregue en **AWS Lambda** dunha función escrita en **Python** que implemente un **bot de Twitter** que use datos de **Spotify**. O funcionamento do mesmo será simple, usaremos o servizo AWS Lambda para a execución da nosa función en base a un evento tipo **cron**, que como explicamos anteriormente, consistirá na execución do código subido a Lambda cada certo tempo X personalizado, no noso caso, todos os domingos do ano ás 21:00.

O **obxectivo** desta parte práctica será poder crear unha conta de Twitter que automaticamente, cada semana, publique un ou varios tweets cas cancións engadidas a unha playlist persoal de Spotify nos últimos 7 días, imitando así, o funcionamento dun *detector* de cancións favoritas do usuario durante todo o ano, que ademais, publique os resultados acompañados de diversos links nunha conta de Twitter para poder consultala en calquera momento e levar un histórico. Por riba de todo isto, a chave será facelo de forma *serverless* e sen ningún tipo de preocupación de administración para o programador, que só se encargará de facer o código e configuración que amosaremos a continuación, para poñelo en marcha e despreocuparse da súa posterior xestión, que como vimos na parte teórica, era o principal beneficio de AWS Lambda.

Polo que o **éxito** do proxecto pasará por aprender a usar as *APIs* de Spotify e Twitter para Python (*Tweepy* e *Spotipy*), para en primeiro lugar, ver qué cancións foron engadidas á playlist durante a última semana, e posteriormente publicar un tweet xeral con toda a información e outro cun *link* por cada unha delas. Para posteriormente, subir todo isto a AWS Lambda e familiarizarse con este servizo e sacarlle o máximo partido, para executar así de forma automática, periódica e *serverless* o código previamente mencionado, simulando un *trackeador* persoal de pistas favoritas durante todo o ano.

Comporase pois de varias **fases** o noso proxecto, as cales serán: a obtención de chaves de desenvolvedor tanto de Twitter como de Spotify, a elaboración do *script* en Python que cumpra ditos requisitos e a subida dos ficheiros a Amazon AWS Lambda para a súa posterior correcta configuración e execución periódica. Faremos pois unha sección por cada unha destas partes, aínda que só entraremos en detalle na referente a AWS Lambda, xa que é o contido principal do traballo e o concepto máis cercano á asignatura. Nese capítulo, mostraremos capturas paso a paso de como facer uso desta ferramenta de igual maneira que se fixo durante o curso cas prácticas da asignatura e no que respecta ás outras seccións, explicaranse os aspectos fundamentais de cada un delas e mencionaranse de forma xenérica o resto de pasos levados a cabo, podendo acudir a eles en detalle nos seus respectivos apéndices. Por último, amosaráse o funcionamento do proxecto cunha serie de probas de execución levadas a cabo, tanto de forma local como da propia función Lambda, amosando en ambos casos capturas dos seus resultados.

Destacar para finalizar esta introdución ao proxecto, que o titorial de referencia no que me baseei pode atoparse en [6].

3.2 Obtención de chaves

3.2.1 Twitter

O primeiro paso será solicitar o acceso a unha conta de *developer* en Twitter, polo que unha vez teñamos a conta de Twitter do bot creada e configurada, encheremos unha serie de formularios para realizar a solicitude en [7].

Neles teremos que detallar a razón da solicitude, loguearnos ca conta de Twitter do bot a usar, pór uns datos básicos sobre ela e finalmente especificar cas nosas palabras a razón da solicitude e as funcións que levará a cabo o bot.

Unha vez feito esto, xa teremos as nosas chaves e *tokens* listos para comezar implementar código, só nos quedará crear un novo proxecto, editar permisos e rexerar unha serie de chaves. Todo isto e máis podemos velo no apéndice referente a Twitter A.

3.2.2 Spotify

En canto a Spotify, como o proxecto usará unha playlist da miña conta para analizala, teremos que loguearnos ca nosa conta persoal na pantalla de desenvolvedores de spotify [8] para obter as chaves e crear unha aplicación de forma parecida á que se fixo anteriormente co proxecto Twitter, só que desta vez será máis rápido e doado. Poderemos ver as imaxes cos detalles no apéndice B.

3.3 Parte de Pyhton

Temos todo listo pois para comezar ca implementación da lóxica do proxecto, polo que comezaremos baseando a estrutura do noso proxecto na arquitectura predefinida exposta en [9]. Polo que o primeiro que faremos será clonar este repositorio.

Despois como se indica na documentación do proxecto, haberá que crear un entorno virtual, activalo e instalar as dependencias base das que partiremos para usar o noso bot.

Posteriormente e para acabar, crearemos un ficheiro *.env* cas variables de entorno do programa que contarán cos *tokens* solicitados nas dúas anteriores seccións e cas constantes da playlist a analizar, para se nun futuro cambia algún destes valores, tan só cambiar aquí a variable e non tocar nada do código. Unha vez que temos esto, procederemos a editar o ficheiro chave do directorio: **lambda_function.py**, onde estará a funcionalidade do noso programa e o cal será posteriormente o código subido da nosa función Lambda.

Polo que agora o traballo que nos queda é facer código neste ficheiro e *testealo* localmente até que o *script* faga as funcionalidades que queiramos da maneira adecuada. Tras esto, poderemos poñernos por fin, ca parte de AWS Lambda.

Xa que isto non é un obxectivo da asignatura e só consiste en lidiar cas *APIs* de Twitter e Spotify e con Python en xeral, non se vai facer fincapé nesta parte aínda que teremos de igual forma que nos dous casos anteriores, un apéndice que conte con todo o código do noso *script* e capturas referentes a esta sección, concretamente o apéndice C.

Antes de pasar ao siguiente paso teremos que executar os *scripts* dispoñibles no repositorio que nos empaquetarán o código e librerías do noso proxecto para telo listo para subir a AWS Lambda en correspondentes arquivos *.zip*.

3.4 Parte de AWS Lambda

Imos agora, que temos a lóxica do noso proxecto feita e todas as partes empaquetadas e listas para ser usadas por AWS Lambda, a mostrar paso por paso, de forma detallada, todas as accións a facer na consola de AWS Lambda para despregar unha función que execute o noso *script*. Polo que imos comezar: primeiro de todo teremos que crear unha capa ou ***Layer*** en AWS Lambda que conteña as librerías que usará o noso proxecto. Polo que na consola de AWS escribiremos Lambda e entraremos:

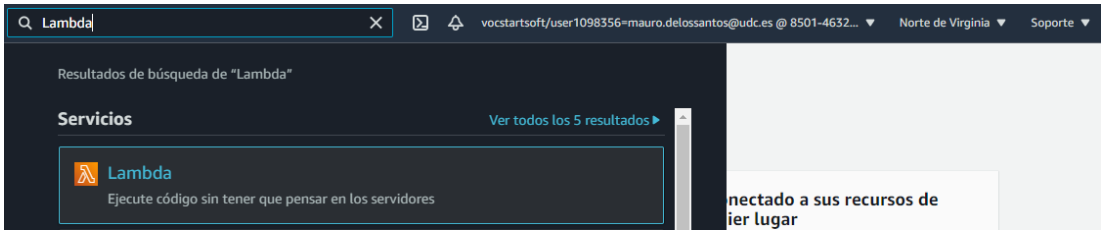

Figura 3.1: Entrando á consola de AWS Lambda

No menú principal iremos na esquerda a Capas->Crear una Capa:

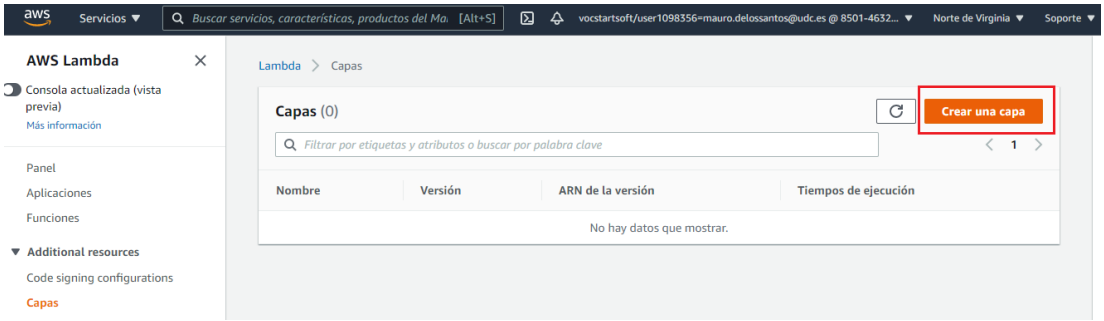

Figura 3.2: Creando a capa

Unha vez aquí, darémoslle nome á nosa capa, subiremos o arquivo *layer.zip* autoxerado cas execucións dos *scripts* anteriorente mencionados e seleccionaremos a versión de *Pyhton* cos cales os executamos, que no noso caso é a 3.7. Finalmente *clickaremos* en crear e esperaremos un anaco ata que a capa se xere correctamente:

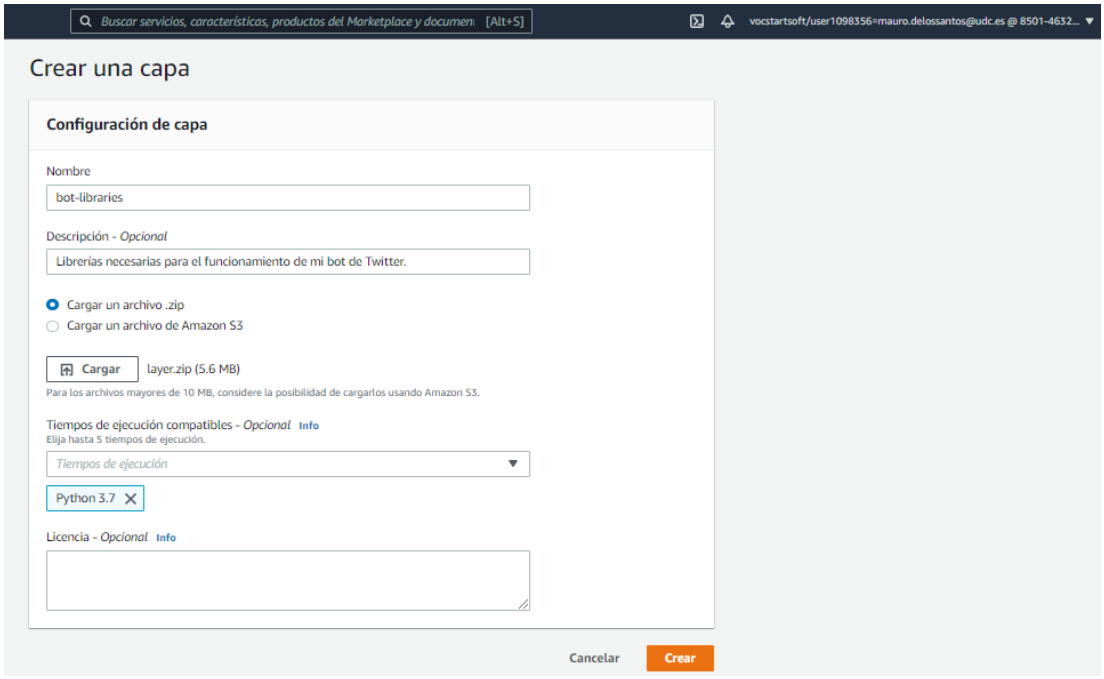

Figura 3.3: Configurando a capa

Con esto, xa temos a nosa capa Lambda creada. Imos agora ca Función Lambda, a lóxica do proxecto. Seleccionaremos Funciones, no menú de AWS Lambda e posteriormente en Crear una Función:

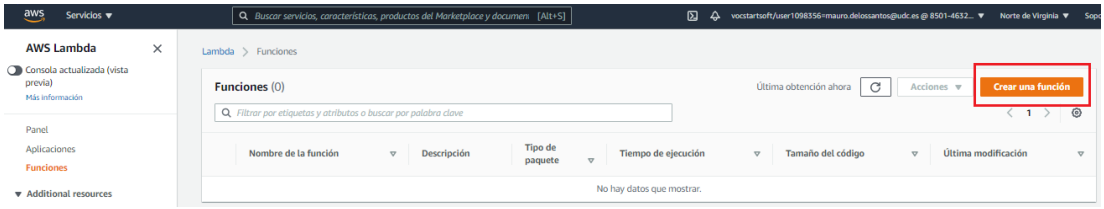

Figura 3.4: Creando a función

Elexiremos un nome e un *runtime* que como ben vimos será *Python 3.7*:

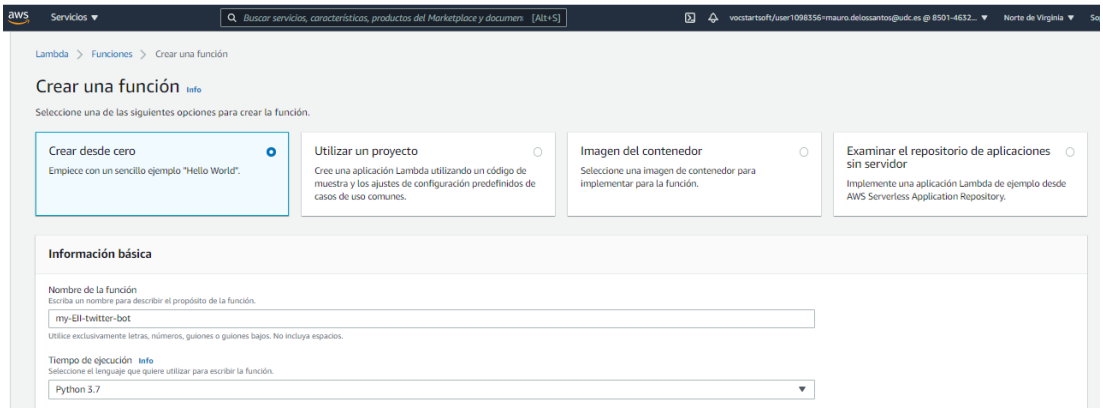

Figura 3.5: Configurando a función (I)

Teremos agora que subir o código que vai executar, para isto iremos a Código de la Función, *clickaremos* en Acción e Subir un ficheiro *.zip*, seleccionando o ***lambda_function.zip*** xerado polos *scripts* executados previamente.

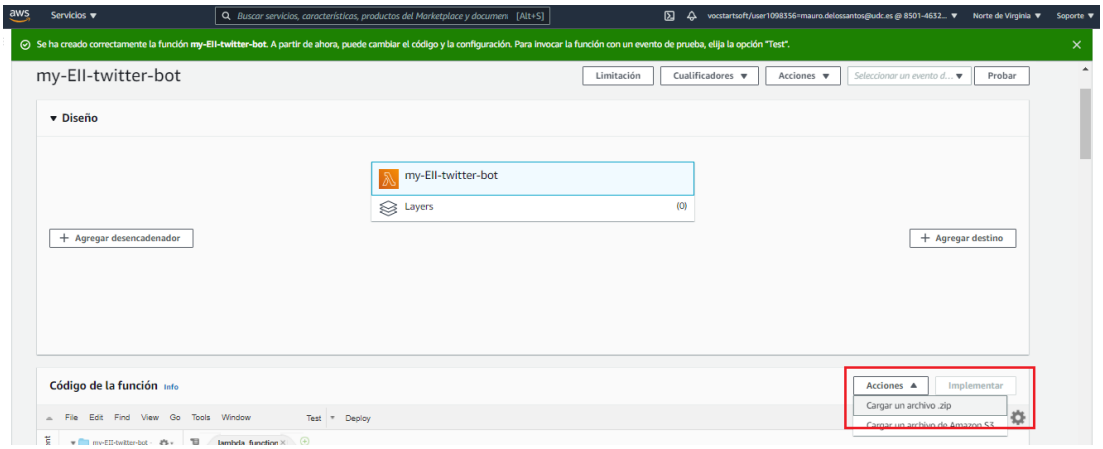

Figura 3.6: Configurando a función (II)

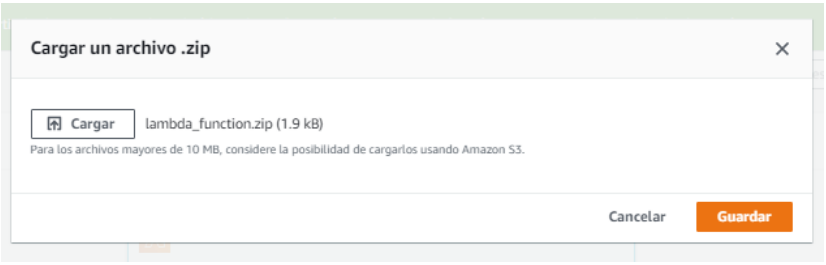

Figura 3.7: Configurando a función (III)

Teremos tamén que engadir as variables de entorno que usemos no noso programa, como son as chaves das *APIs* de Twitter e Spotify e as constantes do nome e identificador da playlist:

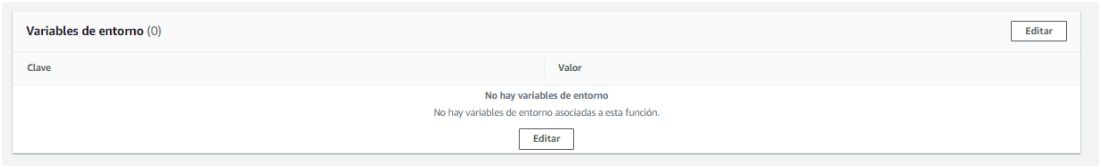

Figura 3.8: Configurando as variables de entorno (I)

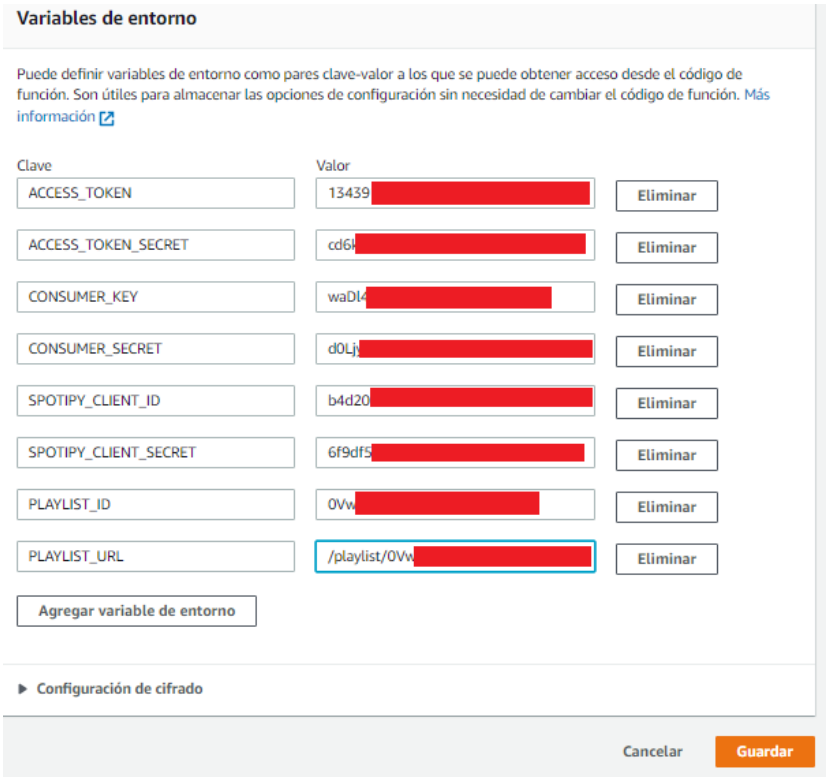

Figura 3.9: Configurando as variables de entorno (II)

Unha vez que o teñamos, teremos que conectar a nosa capa ca nosa función, seleccionando Capas e *clickando* en Añadir una capa:

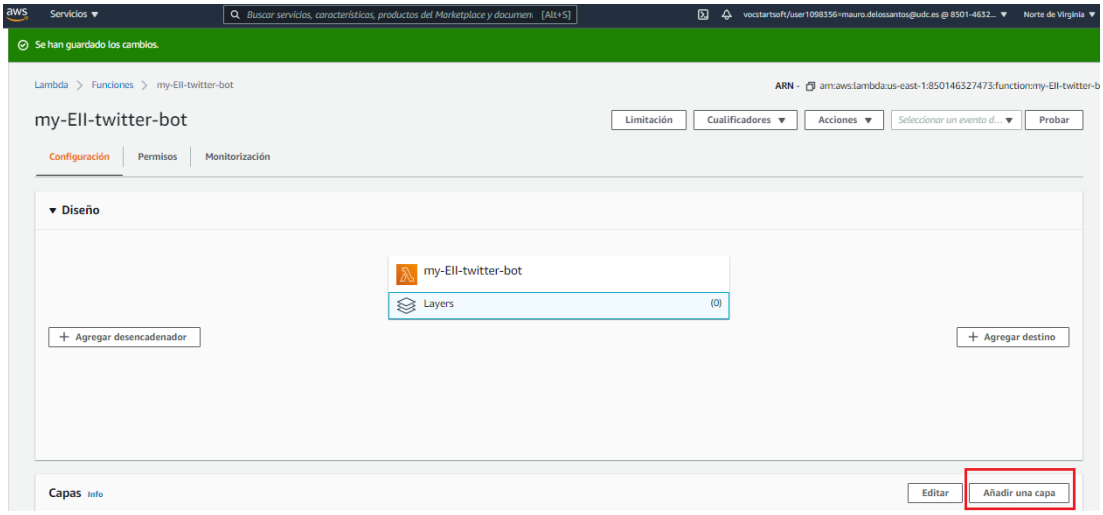

Figura 3.10: Conectando a función ca capa (I)

Seleccionamos Capas personalizadas, eleimos a nosa, o noso número de versión dependendo de cantas veces levemos feito este proceso ou modificado o código da función, e damoslle a Agregar.

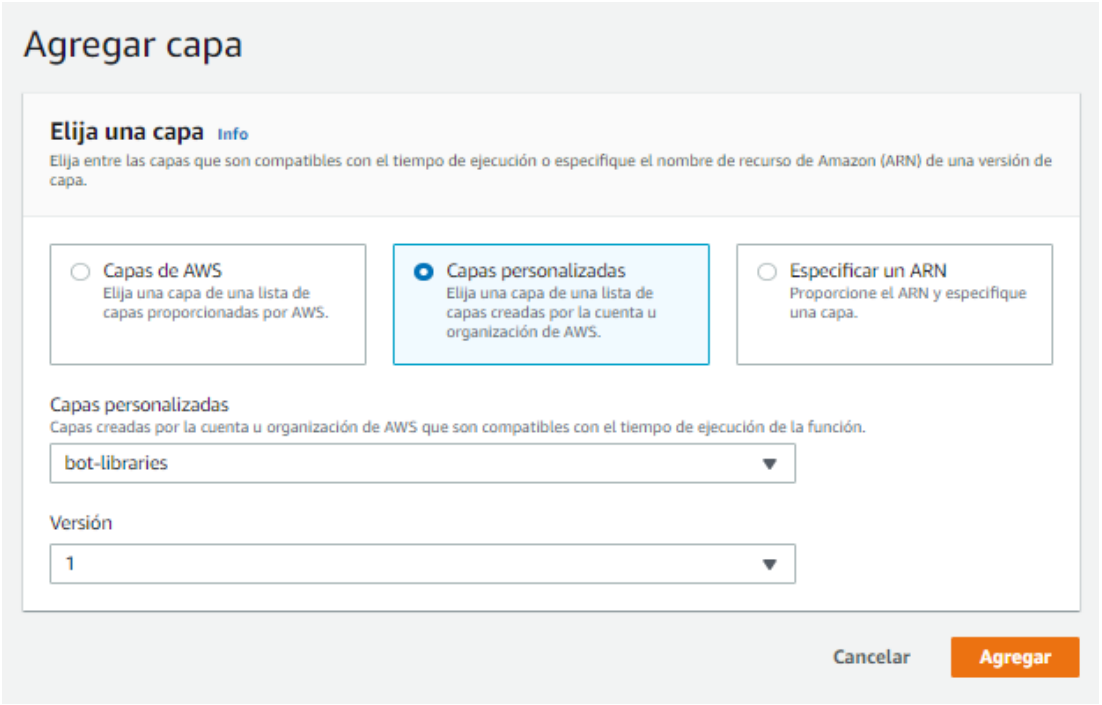

Figura 3.11: Conectando a función ca capa (II)

Con esto, témolo todo listo. Só fará falta crear o *Trigger* que se encargue de executar a nosa función. Esto dependerá de cada caso de uso pero como ben explicamos ao longo desta memoria o obxectivo é mostrar as cancións favoritas durante a última semana, polo que será adecuado executar esta función todos os domingos ás 21:00, por exemplo, polo que procedemos a configuralo. En Configuración *clickamos* en Agregar desencadeador:

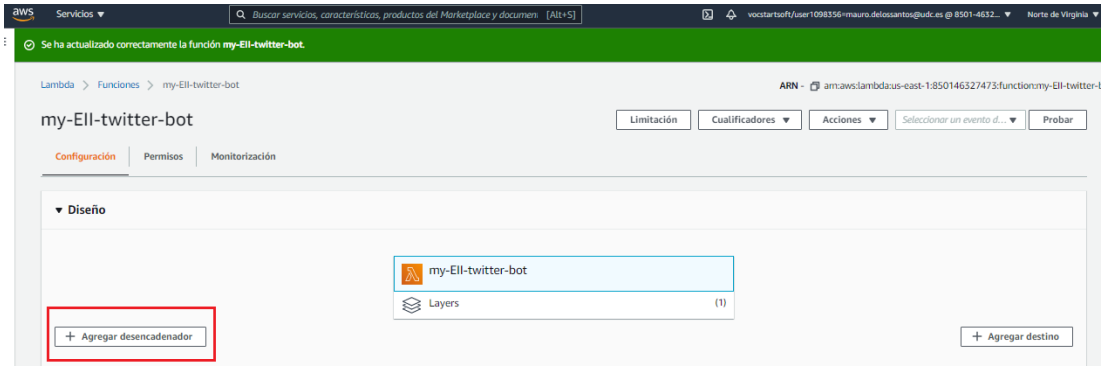

Figura 3.12: Engadindo desencadeador

Creamos unha regla de tipo *EventBridge*, dámoslle nome e definímola cunha Expresión de programación, estilo cron, no noso caso será do tipo: **cron(0 21 ? * SUN *)**. Para crear o noso personalizado podemos ver a información necesaria en [10]. Por último activaremos o *trigger* ou desencadeador:

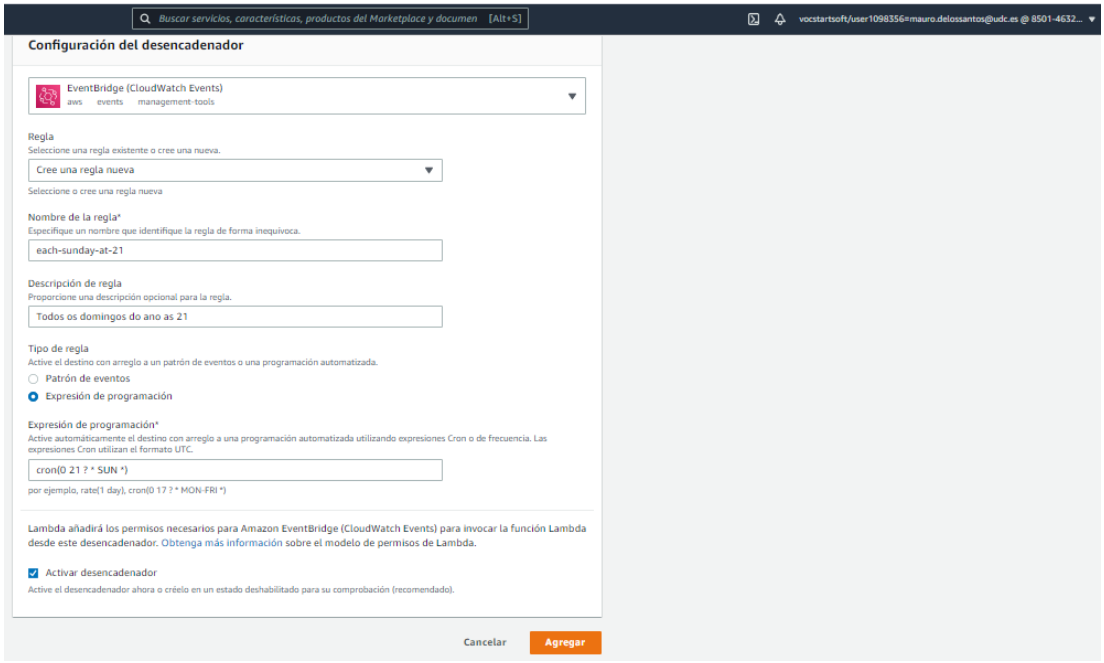

Figura 3.13: Configurando desencadeador

Con esto, acabariamos toda a parte de AWS Lambda, polo que procedemos probar agora toda a infraestrutura e proxecto creado até o momento para ver se o despregue e o funcionamento son da forma esperada.

3.5 Probas e resultados

3.5.1 Local

En canto ás probas realizadas, comentar que foron de diferentes tipos, e que en primeiro lugar, antes de comezar ca parte de AWS Lambda, tívose que executar e probar de forma local o código do *script* **lambda_function.py** que posteriormente se subiría a Lambda mediante a execución do comando:

```
1 $ python3 entrypoint.py
```

Esto fíxose tantas veces como probas necesitamos facer ata conseguir a lóxica esperada. Ao rematar de deseñar o programa, teriamos unha saída do estilo seguinte:

```
(venv) pi@raspberrypi:~/twitter-bot-python-lambda/twitter-bot-python-aws-lambda $ python3 entrypoint.py
Get credentials
Authenticate
Calling spotify function
Calling twitter function
Posting tweet: Esta semana ha sido una locura con 3 semanales! Las dejamos por aquí: Una Cumbia Triste de Rels B, Pum
p It Up de Endor, y Hoy de Jamby el Favo.
Posting tweets with songs links
Posting tweet with playist link
```

Figura 3.14: Probas en local do noso programa

E un resultado en Twitter da seguinte forma:

Figura 3.15: Probas en local do noso programa, parte de Twitter (I)

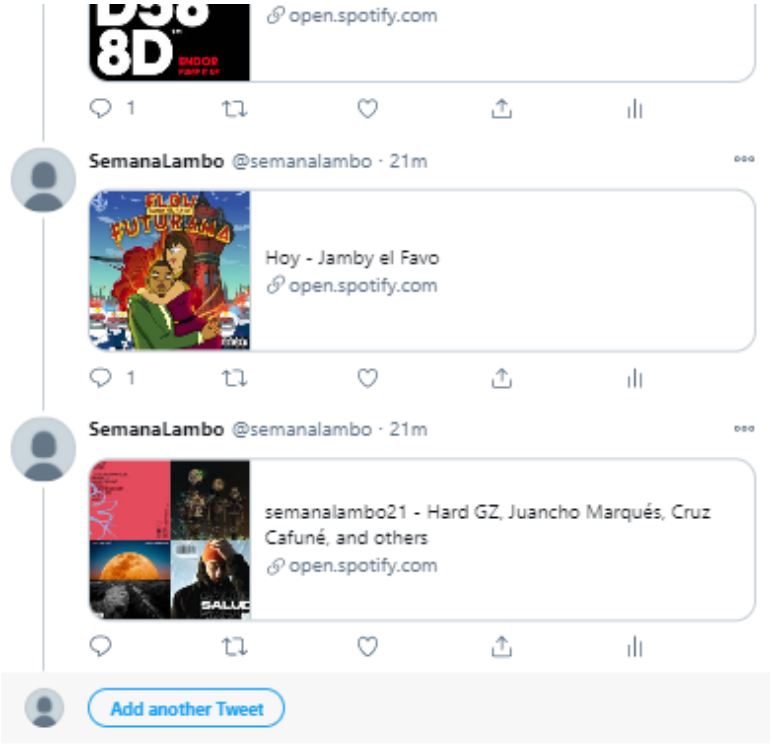

Figura 3.16: Probas en local do noso programa, parte de Twitter (II)

Polo que neste momento, vimos que conseguimos o funcionamento que buscamos na nosa futura función Lambda. É dicir, un *script* que mire na playlist especificada as cancións engadidas nos últimos 7 días e escriba:

- Un fío co tweet principal a modo de resumo cos títulos e autores das cancións.
- Tantos tweets como cancións haxa dita semana, poñendo unicamente o *link* de Spotify de cada unha no propio tweet.
- Un último tweet para pechar o fío co *link* da playlist en Spotify.

3.5.2 Online

Tras subir e configurar o código en AWS Lambda, teremos que *testear* o noso bot *clickando* no botón Test, onde se executará a función Lambda configurada e teremos os resultados e *logs* da mesma para corroborar que a lóxica foi a esperada. Cambiaremos as cancións da playlist para ver se cambia a saída respecto ás probas locais:

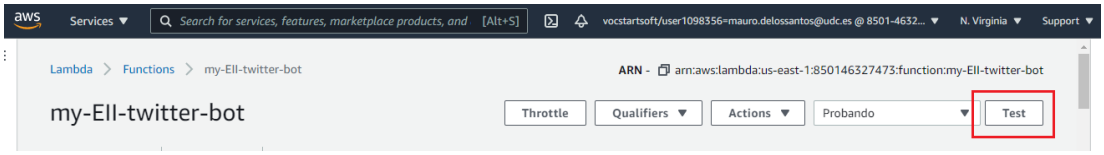

Figura 3.17: Probando o bot en AWS Lambda

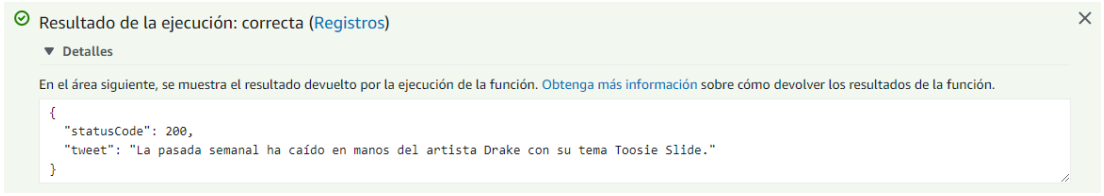

Figura 3.18: Probas en AWS Lambda (I)

Figura 3.19: Probas en AWS Lambda (II)

Figura 3.20: Probas en AWS Lambda, resultados en Twitter

Como vemos, todo saíu da forma que se esperaba. Polo que temos todo preparado, e agora só quedará esperar a que cheguen as nove da noite de cada domingo, e automaticamente Amazon AWS lance un evento e execute esta función Lambda como se foramos nós os que lle estiveramos dando ao botón Test como na proba xustamente anterior, só que ca vantaxe de que non teremos que preocuparnos de ningún tipo de tarefa administrativa, xa que AWS Lambda fará todo por nós. Unicamente, nos adicaremos se quixeramos a mirar os *logs* das execucións e os diversos paneis de monitorización dispoñibles, que ben comentamos na sección teórica e que exemplificamos cas seguintes capturas:

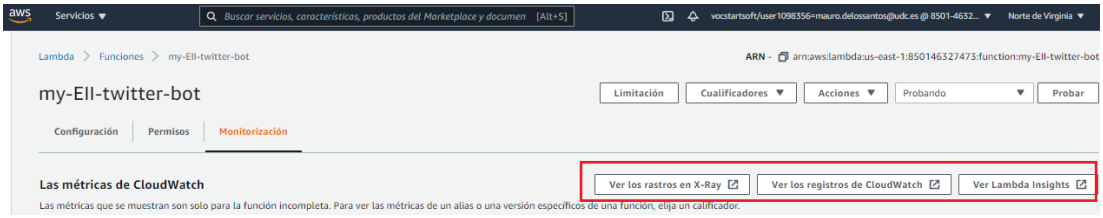

Figura 3.21: Opcións de monitorización en AWS Lambda

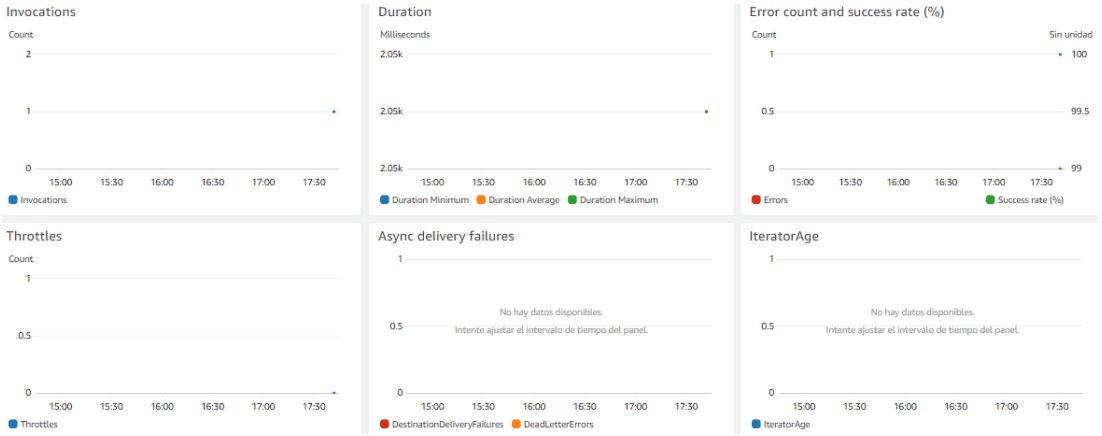

Figura 3.22: Monitorización en AWS Lambda (I)

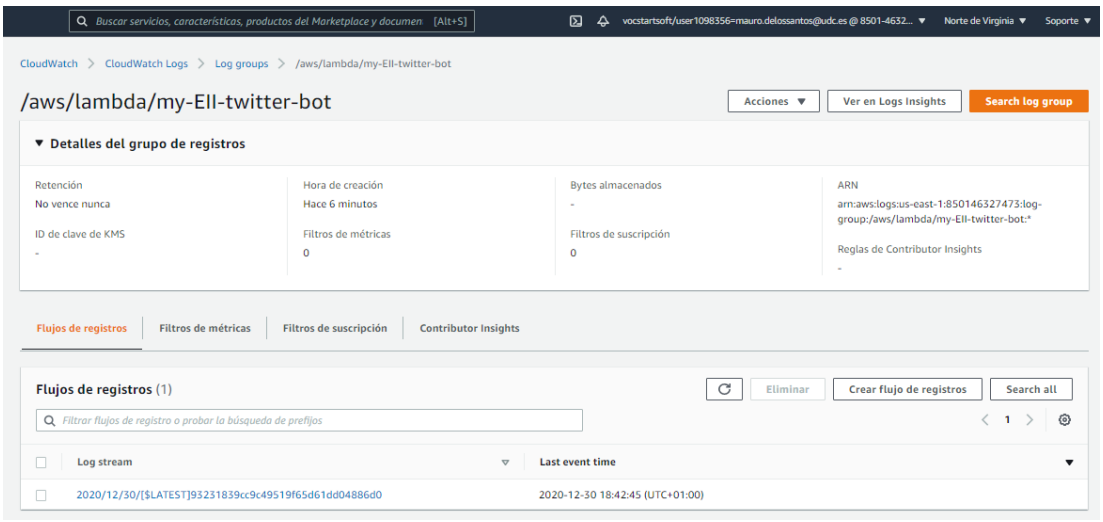

Figura 3.23: Monitorización en AWS Lambda (II)

Capítulo 4

Conclusións

Saco pois en conclusión despois de realizar este traballo, en primeiro lugar, a gran **utilidade e capacidade** que presenta AWS Lambda, servizo descoñecido por min antes de comezar este traballo e que sen dúbida utilizarei despois de facelo, debido ás grandes posibilidades que oferta, brindando un servizo que administra, monitoriza e aprovisiona todos os servidores que poida necesitar nun futuro para aloxar proxectos persoais, preocupándome tan só do código que executará o propio Lambda unicamente cando sexa necesario e que tamén me evite as preocupacións polas execucións consumidoras de grandes cantidades de recursos ou polos problemas de infraestrutura.

Por outro lado, tamén vin o crecemento e mellora da oferta de produtos de ***Cloud Computing*** e como cada vez estes son máis accesibles e están máis a man de calquera usuario, mellorando a experiencia do cliente non profesional cuxos pequenos proxectos son baixos en recursos e presuposto.

E por último, dicir que ao investigar xa non sobre AWS Lambda ou AWS en xeral, se non sobre os **servizos na nube** de forma global, decateime da gran cantidade de información e opcións dispoñibles que se poden atopar de primeira man e de forma doada hoxe en día grazas á cal me levo un novo concepto de cando e como usar estas ferramentas, véndoas agora xa non só como útiles para complexos requisitos ou proxectos, se non tamén como ferramentas que poden axudar a calquera usuario con proxectos persoais de calquera tamaño e tipo, coma foi no meu caso, ca parte práctica desta memoria.

Apéndices

Apéndice A

Twitter

Imos amosar a continuación unha serie de imaxes que exemplifiquen os procesos mencionados na sección de solicitude de chaves de desenvolvedor de Twitter:

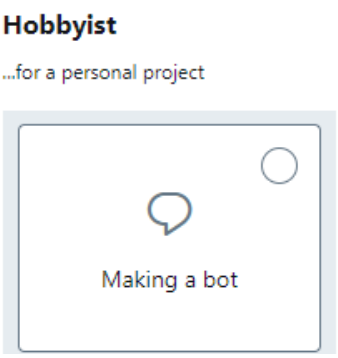

Figura A.1: Explicación do obxectivo da conta de Developer de Twitter

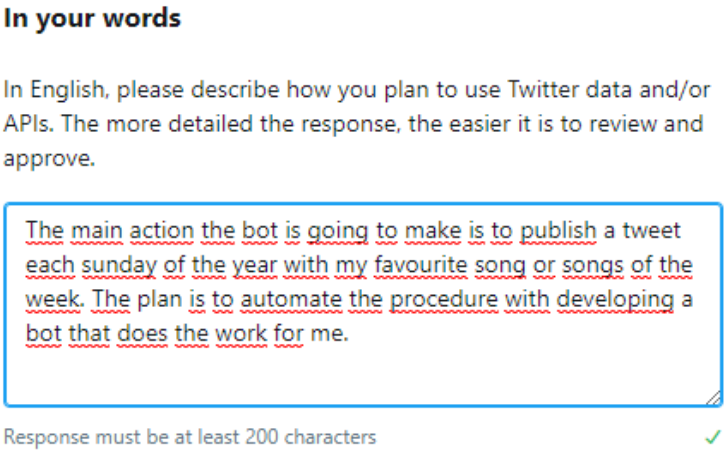

Figura A.2: Descrición cas miñas palabras do funcionamento do meu bot

Figura A.3: Creación do proxecto que usará o bot

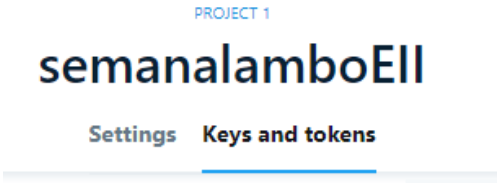

Figura A.4: Apartado de chaves do proxecto

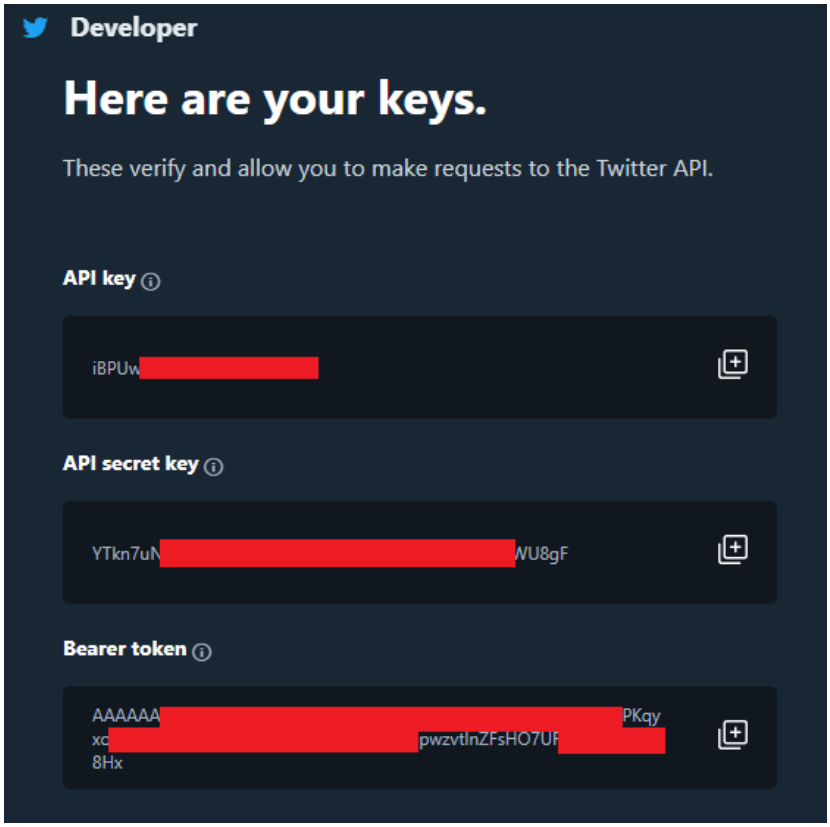

Figura A.5: Chaves de Twitter conseguidas para o proxecto

Figura A.6: Tokens de autenticación Twitter conseguidas para o proxecto

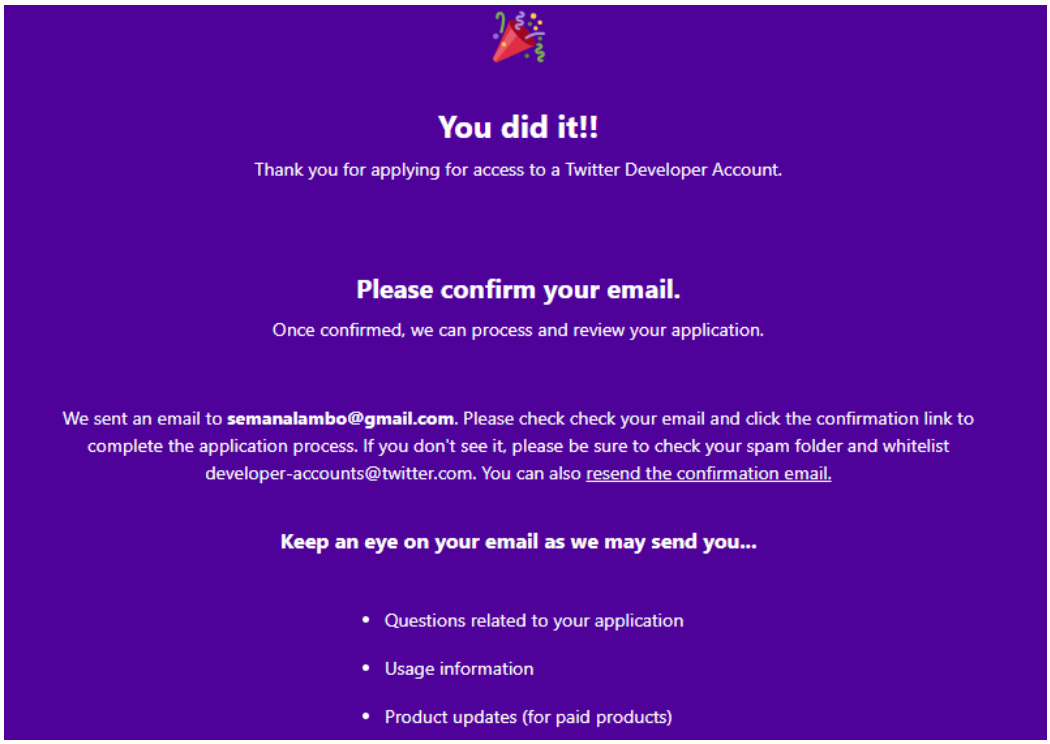

Figura A.7: Confirmación de que todo foi ben

Apéndice B

Spotify

O mesmo para Spotify:

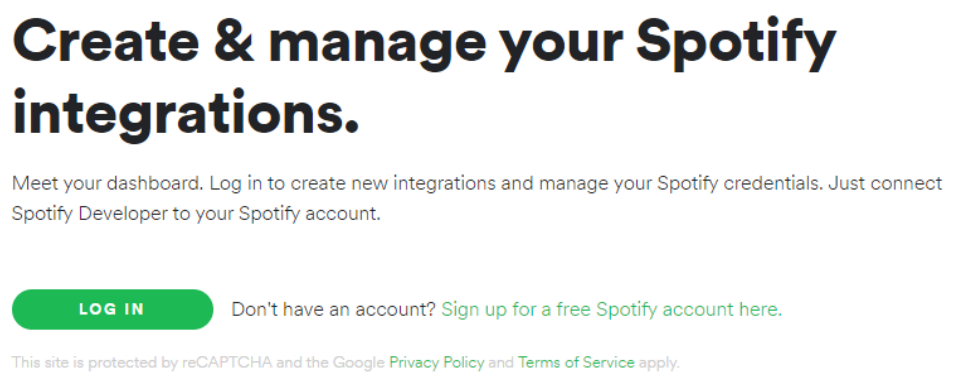

Figura B.1: Logueándonos ca nosa conta para activar as opcións de Developer

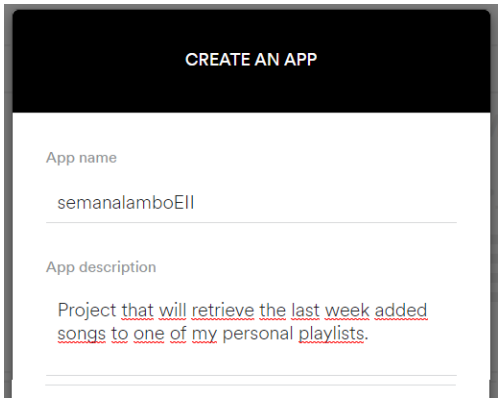

Figura B.2: Creando a nosa app

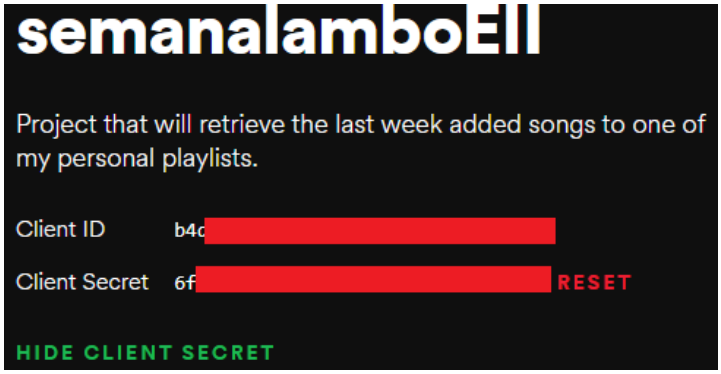

Figura B.3: Obtención das chaves tras crear a nosa app

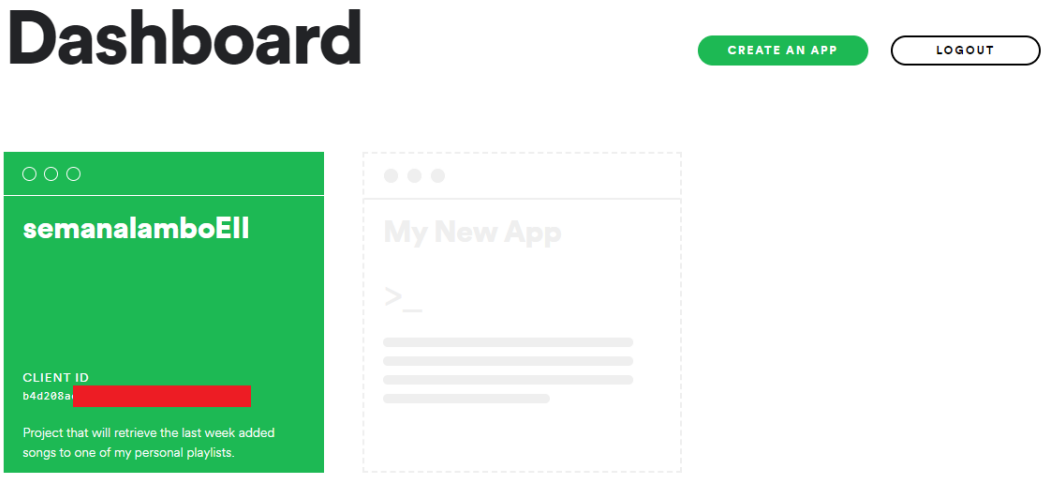

Figura B.4: Dashboard nas opcións de Developer de Spotify

Apéndice C

Python

Comandos previos a comezar facer código:

```
$ python3 -m venv venv
$ source venv/bin/activate
(venv) $ pip install -r requirements.txt
```

```
ACCESS_TOKEN=1343910873[redacted]
ACCESS_TOKEN_SECRET=cd6kSZa1[redacted]
CONSUMER_KEY=waDl4JYD[redacted]
CONSUMER_SECRET=d0LjyXL5OK2[redacted]
SPOTIPY_CLIENT_ID=b4d2[redacted]
SPOTIPY_CLIENT_SECRET=6f9df5f07e10498[redacted]
PLAYLIST_ID=0VwKbO7[redacted]
PLAYLIST_URL=https://open.spotify.com/playlist/0VwKbO7sCr2pAzA2YN3WHY
```

Figura C.1: Contido do ficheiro .env

Código da lóxica do noso proxecto en **lambda_function.py**:

```
import os
import json
import tweepy
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from datetime import datetime, timedelta

def date_comparator(date):
    week_ago = datetime.now() - timedelta(days=7)

    full_date = date.split('Z'); #quitamos a Z final
    full_date[0] = full_date[0].replace('T', '-') #cambiamos a T
    por - para parsear mellor

```

```
14      date_song_added = datetime.strptime(full_date[0],
        "%Y-%m-%d-%H:%M:%S") #creamos a data
15
16      return date_song_added > week_ago
17
18  #devolve autores links e cancions engadidas a playlist na ult semana
19  def get_spotify_info(sp):
20      playlist_id = os.getenv("PLAYLIST_ID")
21      #print(sp.user_playlists('maurocsp98'))
22      tracks_in_playlist =
        sp.user_playlist_tracks('maurocsp98',playlist_id)
23      tracks=[]
24      artists=[]
25      links=[]
26
27      for j in range(len(tracks_in_playlist['items'])):
28          #se a cancion se engadiu na ult semana
29
        if(date_comparator(tracks_in_playlist['items'][j]['added_at'])):
30
        tracks.append(tracks_in_playlist['items'][j]['track']['name'])
31              artists.append(tracks_in_playlist['items'][j]['track']
             ['artists'][0]['name'])
32              links.append(tracks_in_playlist['items'][j]['track']
        ['external_urls']['spotify'])
33      return [tracks, artists, links]
34
35  #escribe o tweet principal a publicar
36  def get_tweet(tracks,artists):
37      tweet = ""
38      semana = str(datetime.now().isocalendar()[1]%52)
39      if (len(tracks) > 2):
40          tweet = 'Esta [' + semana + 'ª] semana ha sido una locura
        con ' + str(len(tracks)) + ' semanales! Las dejamos por aquí: '
41          for x in range(0,len(tracks)):
42              if (x==(len(tracks)-1)):
43                  tweet+= "y " +tracks[x]+" de " + artists[x]+"."
44              else:
45                  tweet+= tracks[x]+ " de " + artists[x] + ", "
46      elif (len(tracks) > 1):
47          tweet = 'En la semana número [' + semana +'] del año hemos
        tenido ' + str(len(tracks)) + ' semanales. Y han sido: '
48          for x in range(0,len(tracks)):
49              if (x==(len(tracks)-1)):
50                  tweet+= tracks[x]+" de " + artists[x]+"."
51              #alternamos 'junto con' e 'y' para non facer repetitivo
```

```
    o tweet
52              elif((x % 2 == False)):
53                  tweet+= tracks[x]+ " de " + artists[x]+ " junto con
    "
54              else:
55                  tweet+= tracks[x]+ " de " + artists[x]+ " y "
56      elif (len(tracks) == 0):
57          tweet = "Lamentablemente esta [" + semana + "ª] semana del
    año no ha habido ninguna canción semanal. Volveremos con más en
    las siguientes semanas!"
58          tweet += " Aún así, puedes echarle igualmente un vistazo a
    la playlist por aquí debajo: "
59      else: #solo 1 cancion
60          tweet = "La semanal nº  " + semana + " del año ha caído en
    manos del artista " + artists[0] + " con su tema " + tracks[0] +
    "."
61      return tweet
62
63  def lambda_handler(event, context):
64      #chaves no ficheiro .env
65      print("Get credentials")
66      consumer_key = os.getenv("CONSUMER_KEY")
67      consumer_secret = os.getenv("CONSUMER_SECRET")
68      access_token = os.getenv("ACCESS_TOKEN")
69      access_token_secret = os.getenv("ACCESS_TOKEN_SECRET")
70
71      print("Authenticate")
72      #twitter
73      auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
74      auth.set_access_token(access_token, access_token_secret)
75      api = tweepy.API(auth)
76      #spotify
77      client_credentials_manager = SpotifyClientCredentials()
78      sp = spotipy.Spotify(client_credentials_manager=
    client_credentials_manager)
79
80      print("Calling spotify function");
81      spotify_info = get_spotify_info(sp)
82      tracks = spotify_info[0]
83      artists = spotify_info[1]
84      links = spotify_info[2]
85
86      print("Calling twitter function")
87      tweet = get_tweet(tracks,artists)
88
89      print("Posting tweet: " + tweet)
```

```
    #vamos comprobar os caracteres do tweet
    if (len(tweet)<=280):
        api.update_status(tweet)
    elif (len(tweet)<=560):
        tweet, tw1 = tweet[:276], tweet[276:]
        api.update_status(tweet+" (+)")
        status = api.user_timeline(count=1)
        api.update_status("(+) "+tw1,
    in_reply_to_status_id=status[0].id)
    else:
        long_tweet = tweet.split('!');
        api.update_status(long_tweet[0]+"! Os dejamos los links por
    aquí debajo:")

    print("Posting tweets with songs links")
    #pon un tweeet respondendo ao tweet anterior por cada cancion
    semanal co seu link
    #para crear un fío dende o tweet principal
    for link in range(0,len(links)):
        api.update_status(links[link],
    in_reply_to_status_id=api.user_timeline(count=1)[0].id)

    print("Posting tweet with playist link")
    #colle o ultimo tweet publicado no perfil
    status = api.user_timeline(count=1)
    playlist_url = os.getenv("PLAYLIST_URL")
    #ao final do fío pon un tweet co link da playlist
    api.update_status(playlist_url,
    in_reply_to_status_id=status[0].id)

    return {"statusCode": 200, "tweet": tweet}
```

Últimos comandos antes de comezar ca parte de AWS Lambda:

```
# sh createambdalayer.sh 3.7
# sh buildapackage.sh
```

Glosario

Amazon API Gateway Servizo de AWS para a creación, publicación, mantemento, monitoreo e protección de API REST, HTTP e WebSocket a calquera escala. 4

Amazon Linux AMI Imaxe de Linux mantida e compatible que oferta Amazon Web Services. 5

Amazon S3 buckets Servizo de almacenamento de obxectos creados para almacenar e recuperar calquera volume de datos dende calquera ubicación de Internet. 3

CloudWatch Servizo de monitorización e observación de AWS. Recopila datos de monitorización e operacións en formato de rexistros, métricas ou eventos, o cal ofrece unha vista unificada dos recursos, aplicacións e servizos de AWS que se están a executar. 7

Cron Utilidade de Linux que establece un comando ou secuencia de comandos para que certas tarefas sexan executadas automaticamente a unha hora e data especificadas. 4

EventBridges Servizo de bus de eventos sen servidor que facilita a conexión de aplicacións con datos de diversos orixes. Anteriormente eran os Amazon CloudWatch Events. 3

Bibliografía

[1] "Aws lambda - guía para desarrolladores," https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html, accedido en decembro de 2020.

[2] "In depth aws lambda overview," https://medium.com/faun/in-depth-aws-lambda-overview-1eeb4580696b, accedido en decembro de 2020.

[3] "Aws lambda," https://aws.amazon.com/es/lambda/, accedido en decembro de 2020.

[4] "Tiempos de ejecución de aws lambda." https://docs.aws.amazon.com/es_es/lambda/latest/dg/lambda-runtimes.html, accedido en decembro de 2020.

[5] "¿qué es aws lambda?" https://serverless-stack.com/chapters/es/what-is-aws-lambda.html, accedido en decembro de 2020.

[6] "How to make a twitter bot with python," https://towardsdatascience.com/how-to-make-a-twitter-bot-for-free-6bca8298f1ef, accedido en decembro de 2020.

[7] "Twitter developer section," https://developer.twitter.com/en/apply-for-access, accedido en decembro de 2020.

[8] "Spotify developer section," https://developer.spotify.com/dashboard/applications, accedido en decembro de 2020.

[9] "Repositorio ca estrutura inicial do noso proxecto," https://github.com/dylanjcastillo/twitter-bot-python-aws-lambda.git, accedido en decembro de 2020.

[10] "Schedule expressions for rules," https://docs.aws.amazon.com/eventbridge/latest/userguide/scheduled-events.html, accedido en decembro de 2020.