UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aplicación Serverless de Procesado de Datos en AWS

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Vidal González, Jaume

Tutor/a: Moltó Martínez, Germán

CURSO ACADÉMICO: 2021/2022

Resum

Un comerç de patinets elèctrics que compta amb 10 botigues físiques i una web per a les vendes en línia desitja migrar totes les seues dades, que actualment té en una infraestructura *on-premise,* al núvol, ja que volen unificar les dades de totes les seues botigues per a accedir a ells des de qualsevol part i realitzar informes per a cada botiga amb aquestes dades.

Aquestes dades consisteixen en l'estoc actual, les vendes de les botigues i les visites web. També s'utilitzen tres taules mestres amb la informació de cada producte, de cada empleat i de cada botiga física.

Per a això, es planteja el disseny d'una arquitectura serverless en AWS que permeta migrar aquestes dades al núvol i automatitzar l'extracció mitjançant Amazon Kinesis per a obtindre les dades en temps real, l'emmagatzematge mitjançant Amazon S3 i DynamoDB, el processament mitjançant Amazon Glue per a crear prediccions de vendes i l'explotació d'aquestes dades mitjançant QuickSight per a poder generar panells en els quals visualitzar les dades.

Aquest treball pretén dissenyar i implementar aquesta arquitectura realitzant, posteriorment, una anàlisi de cost de la solució mitjançant la ferramenta AWS Pricing Calculator per a aconseguir la migració amb el menor cost possible.

Paraules clau: Computació en el Núvol, *Serverless*, *ETL*, Amazon Web Services.

Resumen

Un comercio de patinetes eléctricos que cuenta con 10 tiendas físicas y una web para las ventas online desea migrar todos sus datos, que actualmente tiene en una infraestructura on-premise, a la nube, ya que quieren unificar los datos de todas sus tiendas para acceder a ellos desde cualquier parte y realizar informes para cada tienda con estos datos.

Estos datos consisten en el stock actual, las ventas de las tiendas y las visitas web. También se utilizan tres tablas maestras con la información de cada producto, de cada empleado y de cada tienda física.

Para esto, se plantea el diseño de una arquitectura serverless en AWS que permita migrar estos datos a la nube y automatizar la extracción mediante Amazon Kinesis para obtener los datos en tiempo real, el almacenamiento mediante Amazon S3 y DynamoDB, el procesamiento mediante Amazon Glue para crear predicciones de stock y ventas y la explotación de dichos datos mediante QuickSight para poder generar paneles en los que visualizar los datos.

Este trabajo pretende diseñar e implementar esta arquitectura realizando, posteriormente, un análisis de coste de la solución mediante la herramienta AWS Pricing Calculator para lograr la migración con el menor coste posible.

Palabras clave: Computación en la nube, *Serverless*, *ETL*, Amazon Web Services.

Abstract

An electric scooter retailer that has 10 physical stores and a website for online sales wants to migrate all its data, which it currently has in an on-premises infrastructure to the cloud because they want to unify the data of all its stores to access them from anywhere and make reports for each store with this data.

This data consists of the current stock, store sales and web visits. Three master tables are also used with the information of each product, each employee and each physical store.

For this, we propose the design of a serverless architecture in AWS to migrate this data to the cloud and automate the extraction using Amazon Kinesis to obtain the data in real time, the storage using Amazon S3 and DynamoDB, the processing using Amazon Glue to create sales predictions and the exploitation of this data using QuickSight to generate dashboards to visualize the data.

This work aims to design and implement this architecture and then perform a cost analysis of the solution using the AWS Pricing Calculator tool to achieve the migration with the lowest possible cost.

Keywords: Cloud computing, *Serverless*, *ETL*, Amazon Web Services.

Índice de contenido

Índice de imágenes

1. Introducción

En la actualidad se puede afirmar que uno de los bienes más preciados para las empresas son los datos. En los últimos años ha habido un gran incremento en el área del procesamiento y tratamiento de datos ya que se ha demostrado que haciendo uso de estos datos se podían obtener mayores beneficios y una mejor toma de decisiones.

Esto llevó a que muchas empresas crearan departamentos enteros e invirtieran grandes sumas de dinero en infraestructura para el procesamiento de datos. Esta inversión no estaba al alcance de todas las empresas y ahí es donde entra en juego la computación en la nube.

En este proyecto podemos apreciar un caso de uso en el que mediante un proveedor cloud obtenemos una arquitectura adecuada para el tratamiento y procesado de datos sin tener que realizar una instalación de un equipo local donde almacenar estos datos y no tener que hacer un desembolso económico tan grande.

Este proyecto se ha realizado en una empresa tecnológica donde se busca solucionar la problemática del cliente y sus necesidades, que son en este caso diseñar e implementar una solución para poder gestionar todos los datos de su comercio de una forma más eficiente.

1.1 Motivación

Desde que empecé el grado de Ciencia de Datos siempre tuve en mente que para elegir el tema del Trabajo Final de Grado me acabaría decantando por el área de modelado y *machine learning* (aprendizaje automático) ya que me parecía el campo con mayor potencial.

No fue hasta tercero de carrera que cambió mi perspectiva de una manera notoria debido a dos asignaturas concretas. La primera fue Visualización, donde descubrí que lo que más me llamaba la atención de la ciencia de datos era su última capa, la interpretación y visualización de los resultados. Por esto, tenía decidido hacer un TFG en el que desarrollaría una aplicación web en la que primero se realizaría una extracción de datos de diferentes webs deportivas para que posteriormente cualquiera pudiera visualizar dichos datos de una forma bonita, sencilla y clara.

La segunda asignatura que cambió mi horizonte fue Infraestructura para el Procesamiento de Datos, concretamente la segunda parte de esta asignatura que consistía en infraestructura cloud. Fue en esta asignatura impartida por mi profesor, y ahora tutor, Germán, donde descubrí por primera vez Amazon Web Services. Enseguida quedé fascinado por la simpleza con la que podías desplegar aplicaciones sin necesidad de que estuvieran en tu equipo local y por la cantidad de servicios que ofrecía dicho operador. Un año más tarde, realizando las prácticas de empresa profundicé más en el

campo del cloud computing de la mano otra vez de AWS y es cuando decidí realizar mi Trabajo Final de Grado sobre este tema.

1.2 Objetivos

El principal objetivo de este proyecto es diseñar y desarrollar una arquitectura serverless cloud que permita el procesamiento de datos de un comercio. Para cumplir con el objetivo principal y establecer un punto de partida, dividiremos éste en los siguientes subobjetivos:

1. Diseñar una capa de ingesta de datos para extraer la información de las tiendas físicas y de la página web

2. Diseñar una capa de almacenamiento donde poder almacenar, ordenar y hacer copias de seguridad de los datos

3. Diseñar una capa de procesamiento donde poder aplicar transformaciones y modelos a los datos para posteriormente obtener conocimiento de dichos datos.

4. Diseñar una capa de explotación en la que, gracias al procesamiento previo, se pueda visualizar claramente los elementos clave a tener en cuenta para la empresa que solicita la aplicación.

5. Hacer un estudio y comparación de todos los servicios del proveedor cloud para analizar cuales se adecuan más a nuestro caso en temas de volumetría y coste.

1.3 Impacto esperado

La realización y publicación de este proyecto tiene diversas ventajas como la democratización del despliegue de aplicaciones. Esto, como se menciona más detalladamente en el Anexo 8.1, se relaciona de gran medida con un Objetivo de Desarrollo Sostenible como es la reducción de las desigualdades ya que cualquiera puede hacer uso de los proveedores cloud para evitar afrontar la gran inversión inicial que significa instalar tu infraestructura local.

Otro impacto esperado podría ser que al adoptar más gente el uso del cloud computing, el consumo energético se vería reducido al estar los datacenters de los proveedores cloud mejor optimizados y alimentados por energía renovable. Esto es relacionable con otro Objetivo de Desarrollo Sostenible como es la acción por el clima. Dicha relación también se puede ver más detallada en el Anexo 8.1.

1.4 Metodología

Algo imprescindible a la hora de realizar un proyecto es definir desde un inicio la metodología que se va a usar. No hay una sola metodología correcta ya que la elección se debe hacer en función de los requisitos que tengas.

La metodología tradicional más famosa es Waterfall o modelo en cascada. Esta metodología se caracteriza por dividir el proyecto en fases y avanzar secuencialmente tal y como se ve en la Imagen 1. No se puede avanzar de fase hasta que la anterior haya finalizado. Es una metodología tradicional en cuanto a que requiere mucha planificación por adelantado y gestiona el tiempo y el seguimiento teniendo en cuenta que todo va a salir según lo previsto. Si bien esto te permite monitorizar el progreso de una forma más sencilla, la falta de flexibilidad que conlleva también puede tener su riesgo si cualquier parte del proyecto se tuerce.

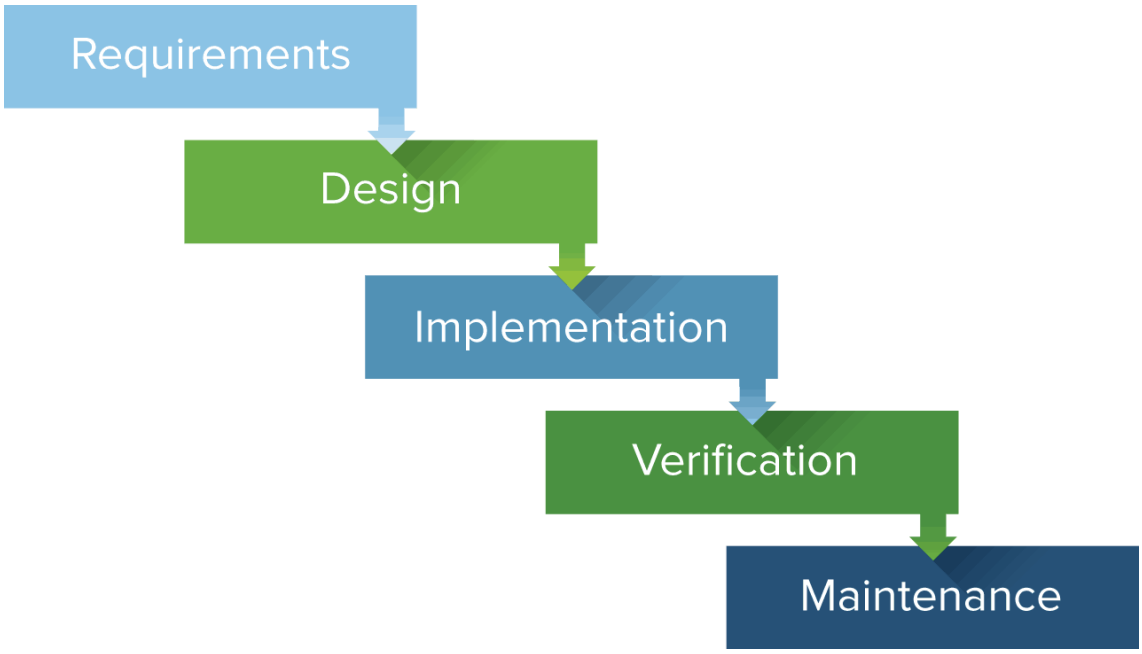

Imagen 1: Fases de la metodología Waterfall [1]

Fue esta falta de flexibilidad la que propició que se desarrollara otra metodología pensada para adaptarse a los cambios inesperados. Esta metodología es conocida como Agile o gestión ágil de proyectos y se ha ido haciendo más relevante en los últimos años. Se caracteriza por ser una alternativa al método tradicional Waterfall y por gestionar las fases del desarrollo de un proyecto de forma simultánea. Esto permite que se puedan hacer distintas fases a la vez mediante lo que se conoce como sprints (véase Imagen 2), que son períodos de trabajo más intensos en los que previamente se han planificado las tareas a realizar.

Imagen 2: Fases de la metodología Agile [2]

Estando este proyecto dividido en fases secuenciales, se ha optado por utilizar la metodología Waterfall ya que se adaptaba mejor a las necesidades. De esta forma, una vez haya finalizado la capa de extracción de datos se podrá pasar a la capa de almacenamiento ya que no tendría sentido hacerlas de forma simultánea al ser necesario que la primera esté completa para poder almacenar los datos posteriormente.

1.5 Estructura de la memoria

Este proyecto se descompone en 8 capítulos que se describen a continuación:

- **Introducción:** En este primer capítulo se realiza una introducción del proyecto donde se expone el problema global que ha llevado a la realización de dicho proyecto y la razón por la cual se ha escogido el tema propuesto. Del mismo modo, se definen los objetivos a seguir y la metodología de gestión de proyectos a utilizar.
- **Estado del arte:** En el segundo capítulo se realiza un análisis de la situación del estado del arte y de la historia del cloud computing.
- **Análisis del problema:** En este capítulo se analiza el problema que ha originado este proyecto, la solución propuesta y sus posibles alternativas y la estimación de los costes de la aplicación.
- **Diseño de la solución:** En el cuarto capítulo se muestra y analiza minuciosamente la arquitectura AWS propuesta. El capítulo está dividido en 5 subapartados que son las propias capas de la arquitectura
- **Implementación y discusión:** En este capítulo se pretende profundizar en la implementación de la arquitectura y analizar y comparar los servicios que hemos utilizado en la arquitectura respecto a otros servicios similares. También se plantea una arquitectura diferente mediante herramientas open source.
- **Conclusiones y trabajos futuros:** En el sexto capítulo se exponen las conclusiones que hemos obtenido con la realización de nuestro proyecto y se

pretende anotar posibles mejoras del proyecto que se podrían implementar en un futuro.

- **Bibliografía:** En el séptimo capítulo se puede observar un listado con todas las referencias que han sido utilizadas para la creación del proyecto.
- **Anexos:** En el octavo y último capítulo se encuentra una reflexión sobre la relación que tiene este proyecto con los Objetivos de Desarrollo Sostenible (ODS) y cuáles son los objetivos más destacables de los que se puede encontrar relación. En otro subapartado de este capítulo encontramos un glosario que facilita la comprensión de términos técnicos utilizados a lo largo de la memoria.

2. Estado del arte

En este segundo capítulo vamos a analizar la historia de la tecnología de computación en la nube y de cómo ha llegado a convertirse en un elemento imprescindible en el panorama tecnológico a dia de hoy.

2.1 Estado del arte

2.1.1 Historia

El origen de la computación en la nube se remonta a la década de los años 60, donde el científico JCR Licklider diseñó ARPANET (Advanced Research Projects Agency Network) [3], una versión precursora de Internet con el objetivo de conectar diferentes ordenadores en diferentes localizaciones para compartir información entre ellos (ver Imagen 3). Posteriormente, en la década de los 70, IBM desarrolló un sistema operativo llamado VM (Virtual Machine) que es el origen de la virtualización.

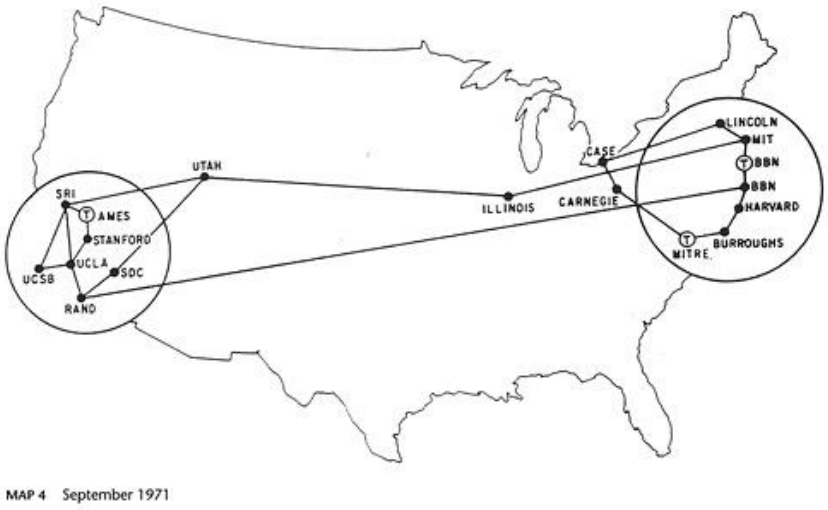

Imagen 3: Mapa de las conexiones ARPANET en 1971 [4]

En los años posteriores hubo un aumento de inversión en datacenters y en el año 2002, Amazon al analizar que tenía mucha capacidad de computación sin utilizar, creó Amazon Web Services (AWS) para incentivar a los desarrolladores a que hicieran sus propias aplicaciones [5].

Este AWS no sería el que conocemos ahora hasta 2006 (ver Imagen 4), donde se lanzarían entre otros los servicios EC2 para la parte de computación y S3 para la parte de almacenamiento. Estos dos servicios fueron y hoy en día siguen siendo dos pilares fundamentales para AWS.

Imagen 4: Logo original AWS [6]

Dos años más tarde, Google también se unió al sector con el lanzamiento de un avance de Google App Engine [7]. Su objetivo era el mismo que el de Amazon en 2002, permitir a desarrolladores crear aplicaciones web en sus servidores pudiendo escalarlas computacionalmente si fuera necesario. No fue hasta finales de 2011 cuando Google lanzó el servicio oficialmente para el público general.

Microsoft anunció su propia plataforma cloud Microsoft Azure en octubre de 2008 pero le llevó hasta febrero de 2010 hacerla disponible al público general añadiendo también el servicio SQL Azure.

Desde el inicio de Amazon en 2006, siempre ha sido la empresa que más dominaba el mercado de la computación en la nube y en la actualidad son estos tres proveedores mencionados anteriormente (Amazon, Google y Microsoft) los que mayor cuota de mercado tienen. AWS ha sido, hasta la fecha, el proveedor cloud más dominante del sector. Está tónica dominante ha empezado a cambiar recientemente, ya que está viéndose amenazada por Microsoft Azure que cada año está consiguiendo incidir más en el mercado. En la Imagen 5 podemos ver la cuota de mercado de diferentes proveedores cloud en el año 2021.

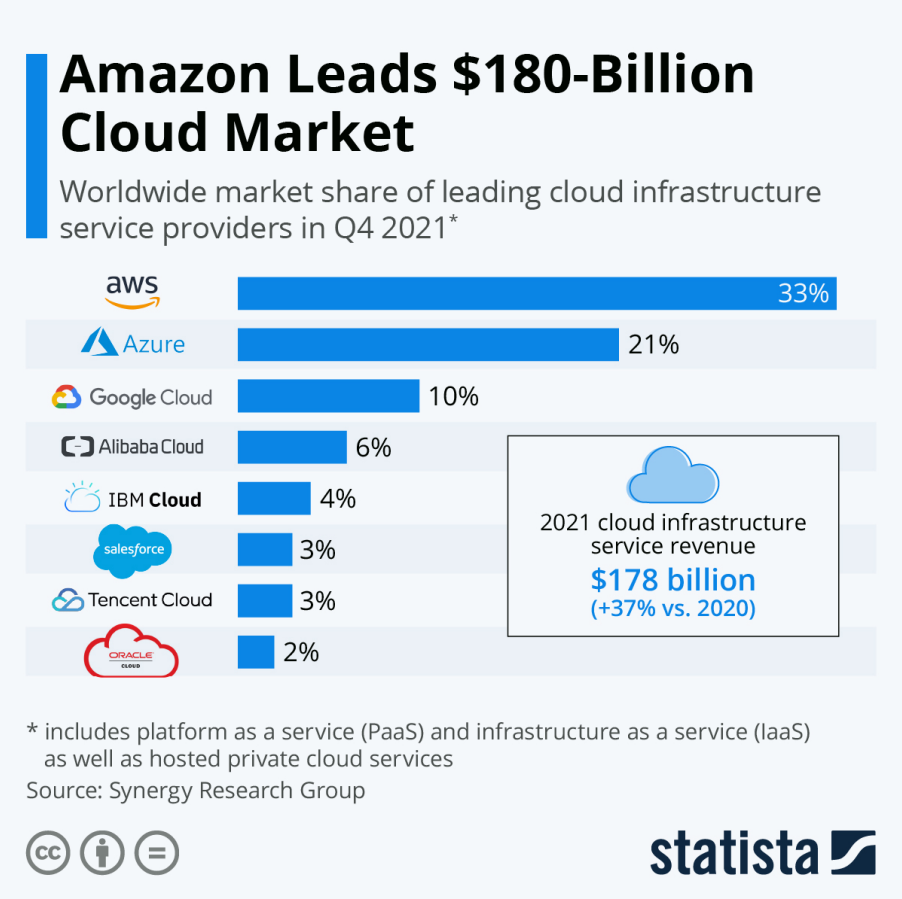

Imagen 5: Cuota de mercado proveedores cloud [8]

2.1.2 Tipos de computación

Una vez hemos adquirido el contexto histórico de la computación en la nube vamos a ver los tres tipos de computación en la nube que existen.

- **IaaS:** IaaS (*Infraestructure-as-a-Service*) es un modelo de infraestructura como servicio. De los tres tipos servicios en la nube es en el que más control tienes de la infraestructura ya que el proveedor se encarga de los servidores, el almacenamiento, la virtualización y la red mientras que el desarrollador se tiene que hacer cargo de elegir sistema operativo y la capacidad de hardware que tendrá la infraestructura como puede ser el número de procesadores, la memoria RAM o el espacio de almacenamiento. Este tipo de infraestructura permite amoldar la configuración a tus necesidades y únicamente pagar por lo que utilices pudiendo detener el servicio en cualquier momento. Los principales proveedores cloud como pueden ser AWS, Microsoft Azure y Google Cloud son claros ejemplos de una infraestructura como servicio.

- **PaaS:** PaaS (*Platform-as-a-Service*) es un modelo de plataforma como servicio. Este modelo está pensado para desarrolladores que quieren implementar una solución rápidamente sin tener que configurar nada ya que es el proveedor cloud

el que se encarga de toda la configuración. Esto puede tener la desventaja de que no puedes elegir la capacidad de computación o el sistema operativo en el que trabajas, pero te permite acceder rápidamente a un entorno donde desarrollar y ejecutar tus aplicaciones sin preocuparte de diseñar la infraestructura. Algunos ejemplos de plataforma como servicio son Heroku, Google App Engine o AWS Elastic Beanstalk.

- **SaaS:** SaaS (*Software-as-a-Service*) es un modelo de software como servicio. En este caso, al ser la capa más superficial del cloud computing, el proveedor se encarga de desarrollar la aplicación y de configurar toda infraestructura necesaria para que el usuario únicamente tenga que conectarse a la aplicación y aprovecharse de sus servicios. La ventaja de este servicio es que cualquier usuario tiene acceso a la aplicación sin necesidad de instalar nada en su ordenador local.

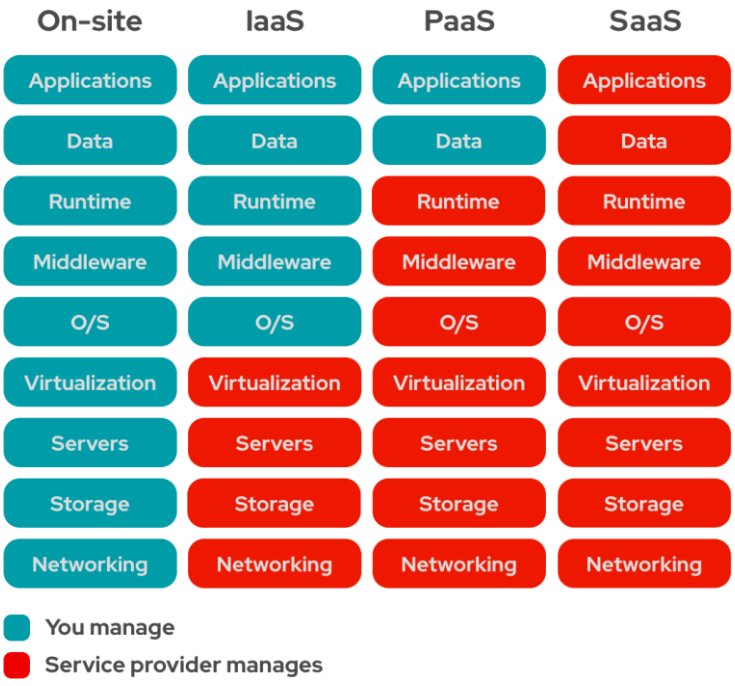

Imagen 6: Tipos de computación en la nube [9]

2.1.3 Características

Para profundizar en el contexto tecnológico, vamos a adquirir en esta sección una vista global de las principales ventajas y funciones que han hecho que se popularice tanto el uso del cloud computing [10]. Las características más destacables son:

- **Precio:** Una de las principales características que más se argumentan para favorecer la migración a la nube es el precio. Esto se debe a que tener una

infraestructura local supone una fuerte inversión inicial y tener un conocimiento técnico para la instalación y el mantenimiento de dicha infraestructura. Con la computación en la nube no te tienes que hacer cargo de esa parte y está establecido el modelo *pay per use* (pago por uso), donde únicamente pagas por los servicios que utilizas y esto supone un gran ahorro cada mes.

- **Seguridad:** Un gran problema al que se enfrentan las empresas actualmente es los continuos ataques informáticos que sufren. Por esto, otra característica a destacar de la computación en la nube es que los proveedores cloud cuentan con expertos que pueden evitar estos ataques teniendo tus aplicaciones más seguras que si las tuvieras en local.

- **Flexibilidad:** Como ya hemos mencionado anteriormente en el apartado de precio, los proveedores cloud cuentan con un modelo pay per use que te permite pagar únicamente por lo que utilizas. Esto sumado a la posibilidad de cambiar la configuración de tu servidor en cualquier momento para, por ejemplo, aumentar la capacidad de procesamiento en un momento de pico de ventas, te añade una flexibilidad que nunca tendrías con un servidor local.

- **Copia de seguridad:** Otra característica que puede resultar muy útil para los negocios es la capacidad de recuperar la información que tienen los proveedores cloud. Al tener sus datacenters en diferentes zonas de disponibilidad, se evitan sufrir posibles desastres naturales, problemas energéticos o averías técnicas por lo que siempre podrás tener acceso a tus datos y no habrá riesgo de pérdida.

- **Accesibilidad:** Una característica especialmente útil en nuestro caso de uso es la accesibilidad a los datos que permite el cloud computing. Esto se debe a que una vez está desplegada la arquitectura, el cliente puede acceder a sus datos desde cualquier parte del mundo únicamente estando conectado a Internet.

2.1.4 Comparación de servicios

Una vez vistas las ventajas que tiene el cloud computing, vamos a agrupar por las principales categorías algunos de los servicios que ofrecen los tres principales proveedores cloud: Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP) [11]:

- **Computación:** En la categoría computación hablamos de servicios que nos permiten ejecutar y desplegar aplicaciones o código. Las máquinas virtuales (VM) son los elementos que más se utilizan de esta categoría. Estas máquinas se pueden configurar para elegir el sistema operativo o los recursos que se le asignarán. En AWS contamos con el servicio Amazon EC2 (*Elastic Compute Cloud*) mientras que en Azure y en GCP con Azure Virtual Machine y Google Compute Engine, respectivamente. Otro servicio muy utilizado en la categoría de computación son las funciones serverless. Estas funciones te permiten

ejecutar código fácilmente en una variedad de lenguajes sin necesidad de indicarle la cantidad de recursos necesaria por parte del usuario.

- **Almacenamiento:** Los principales servicios de almacenamientos suelen ser bases de datos relacionales, bases de datos no relacionales y almacenes de objetos.
  Las bases de datos relacionales en general suelen ser de dos tipos, MySQL o PostgreSQL. Para estas, contamos con Amazon RDS en el caso de AWS, Azure SQL en el caso de Azure y Cloud SQL en el caso de GCP.
  Las bases de datos no relacionales también conocidas como NoSQL (*not only SQL*), son bases de datos más orientadas al rendimiento y a la escalabilidad. Se pueden utilizar con los servicios DynamoDB en AWS, Azure Cosmos DB en Azure y BigTable en GCP.
  Los almacenes de datos son estructuras de datos no jerárquicas en las que se puede almacenar todo tipo de objetos. Se pueden asignar a los objetos prefijos para tenerlos clasificados. En AWS contamos con S3 (*Simple Storage Service*), en Azure con Blob Storage y en GCP con Google Cloud Storage.

- **Big data y machine learning:** También contamos con diversas herramientas para el análisis, procesado y explotación de los datos. Un ejemplo podría ser el flujo de datos en tiempo real, donde se utilizan las herramientas Amazon Kinesis en AWS, Stream Analytics en Azure o Pub/Sub en GCP. También podemos hacer proyectos de machine learning gracias a servicios como SageMaker en AWS, Azure Machine Learning Studio en Azure o Cloud Machine Learning Engine en GCP. Para finalizar el flujo, podríamos visualizar los datos utilizando QuickSight de AWS, PowerBI de Azure o Data Studio de GCP.

2.2 Propuesta

Este trabajo se centra en mostrar un ejemplo de lo que podría ser una solución de cloud computing para un negocio cualquiera. Aplicaciones como ésta se han ideado en muchas ocasiones, pero en este proyecto realizamos una arquitectura similar , pero contando con la novedosa adopción de la tecnología serverless.

Cuando hablamos de serverless, la traducción exacta al castellano significaría "sin servidor”. Esta traducción no sería correcta ya que evidentemente sí que tiene que haber un servidor en algún lugar que ejecute la aplicación. Con serverless nos referimos a recursos completamente administrados por el proveedor cloud.

Haciendo uso de servicios como AWS Lambda, Amazon EventBridge, Amazon API Gateway o Amazon SNS, podemos desplegar una arquitectura con una configuración mínima que escala automáticamente dependiendo de nuestras necesidades y que únicamente se paga por el uso que se le dé. Esto ha supuesto un gran avance respecto a arquitecturas anteriores que dependían más de tener instancias de computación como EC2 desplegadas y suponía una configuración más compleja y un mayor coste.

En la arquitectura propuesta, haremos uso de diferentes servicios de AWS, que serán los siguientes:

- **Amazon Kinesis Data Streams**: Este servicio nos permitirá extraer los datos relacionados con las ventas, las visitas web o el stock en tiempo real.
- **Amazon S3**: Este servicio nos permitirá almacenar los datos independientemente del formato en el que estén.
- **Amazon DynamoDB:** Este servicio nos permitirá almacenar los datos en una base de datos no relacional.
- **Amazon Aurora:** Este servicio nos permitirá almacenar los datos de las tablas maestras en una base de datos relacional MySQL.
- **Amazon SNS:** Este servicio nos permitirá configurar notificaciones para que se disparen cuando sucedan ciertos eventos.
- **AWS Glue:** Este servicio nos permitirá ejecutar fragmentos de código para el procesamiento de datos
- **Amazon SQS:** Este servicio nos permitirá desacoplar eventos entre la API REST y el almacenamiento.
- **Amazon QuickSight:** Este servicio nos permitirá desarrollar un dashboard para visualizar y analizar los datos.
- **Amazon Athena:** Este servicio hará de conector entre Amazon DynamoDB y Amazon QuickSight para poder importar los datos directamente.
- **Amazon API Gateway:** Este servicio nos permitirá extraer o introducir datos en las bases de datos mediante una API REST.
- **AWS IAM:** Este servicio nos permitirá definir usuarios y asignarles roles para tener controlado el acceso a los servicios AWS.
- **AWS CloudWatch:** Este servicio nos permitirá monitorizar los eventos de todos los servicios AWS y poder investigar los posibles errores de éstos.
- **AWS CloudTrail:** Este servicio nos permitirá monitorizar la actividad de los distintos usuarios de la cuenta AWS.
- **AWS Cost Explorer:** Este servicio nos permitirá tener un control de los costes que está teniendo la arquitectura en cualquier momento.

3. Análisis del problema

Este proyecto se originó por la necesidad de una empresa de venta de patinetes eléctricos de procesar toda la cantidad de datos que recibía y utilizarlos para realizar una mejor toma de decisiones. Esta compañía dispone de 10 tiendas físicas con 12 empleados cada una de ellas y un portal web. También dispone de 10 productos diferentes.

La compañía pretende disponer de un control del stock centralizado en tiempo real y disponer de un análisis de las franjas horarias o días con más ventas. Por otra parte, le gustaría disponer de datos de navegación en el portal web para poder analizar que páginas son las más atractivas y evolucionar las que no lo sean tanto para intentar obtener más atracción de clientes. También contempla la posibilidad de compensar a los empleados que consigan más ventas.

Para esto, los directivos de la empresa quieren disponer de un panel administrativo en el que se pueda visualizar el stock por producto en tiempo real, el producto más vendido, el empleado con más ventas semanales, la localización de cada una de las ventas y el análisis del portal web para saber que sección de la web recibe más visitas.

Actualmente la compañía cuenta con una media de ventas diaria de 10.000 patinetes mediante la tienda online y 100 patines por cada tienda física.

La compañía está obligada a mantener los datos de ventas durante 120 días, aunque asume que estos datos sólo serán útiles en el proceso durante 30 días desde su generación.

3.1 Solución propuesta

Para obtener lo deseado por la empresa se le ha propuesto realizar una arquitectura serverless en AWS que permita migrar estos datos a la nube y automatizar la extracción mediante Amazon Kinesis para obtener los datos en tiempo real, el almacenamiento mediante Amazon S3, Amazon Aurora MySQL y DynamoDB, el procesamiento mediante Amazon Glue para crear predicciones de ventas y la explotación de dichos datos mediante QuickSight para poder generar paneles en los que visualizar los datos.

En el diseño de la arquitectura primará escoger los servicios de menor coste que puedan ser funcionales con la volumetría de la empresa.

3.2 Presupuesto

Para hacer un cálculo aproximado del coste que conllevaría la arquitectura que se presenta en este proyecto, hemos hecho uso de la herramienta AWS Pricing Calculator [12] para realizar una estimación del coste total en la región de us-west-2 (Oregón). En la siguiente tabla se puede ver el coste mensual de los servicios de la arquitectura desglosados con una ligera descripción de cada uno.

Servicio	Descripción	Coste mensual
Amazon Kinesis [13] 	Se producen 111.000 registros por día de web y ventas, esto se traduce que cada minuto se extraen 77,08 registros de media.	**11.00 USD**
AWS Lambda [14] 	2 funciones Lambda que leen el Kinesis Streams y escriben los eventos en una DynamoDB. Cada función Lambda se ejecuta, en promedio, 2200 veces al día.	**0.58 USD**
S3 Bucket [15] 	Utilizamos un bucket S3 durante 30 días y luego migramos los datos a Glacier, un sistema de almacenamiento de muy bajo coste, donde deben almacenarse mínimo 90 días.	**0.21 USD**
AWS Lambda 	1 función Lambda que extrae de S3 y almacena los datos de las encuestas a los clientes en DynamoDB. Se ejecuta una vez al día.	**0.01 USD**
AWS Aurora MySQL [16] 	Utilizamos una instancia t3.small y un almacenamiento de 2 GB.	**30.66 USD**
Amazon DynamoDB [17] 	Contamos con un almacenamiento de 3 GB y exportamos 2 GB a S3 cada mes	**1.35 USD**

S3 Backup 	Bucket S3 para guardar las backups de Aurora y DynamoDB durante 30 días y luego pasarán a un S3 Glacier durante 90 días más antes de que se eliminen	**0.70 USD**
AWS Glue [18] 	Tendremos un script de Python que se ejecutará semanalmente.	**0.15 USD**
Amazon Athena [19] 	Consulta diaria de 3 GB cada día para importar los datos de DynamoDB a QuickSight y que estos estén actualizados diariamente.	**0.44 USD**
Amazon Api Gateway [20] 	API REST con 10000 peticiones cada mes	**0.04 USD**
Amazon QuickSight [21] 	Dashboard accesible para las 10 tiendas físicas. Contaremos también con 10 GB de almacenamiento SPICE, que al ser el almacenamiento propio de QuickSight, nos permitirá cargar los datos en el dashboard más rápidamente.	**45.20 USD**

El coste total sería de 90.34 dólares cada mes, haciendo una suma de 1083.96 dólares anuales.

4. Diseño de la solución

En este capítulo vamos a plantear una arquitectura serverless en AWS para resolver el problema comentado en la sección anterior. La arquitectura la vamos a dividir en cinco capas, la cuales tendrán sus propios subapartados.

Las cinco capas serán la capa de ingesta, la capa de almacenamiento, la capa de procesamiento, la capa de explotación y la capa de gobierno. En la Imagen 7 podemos apreciar una vista global de la arquitectura y como interaccionan todos los servicios entre sí.

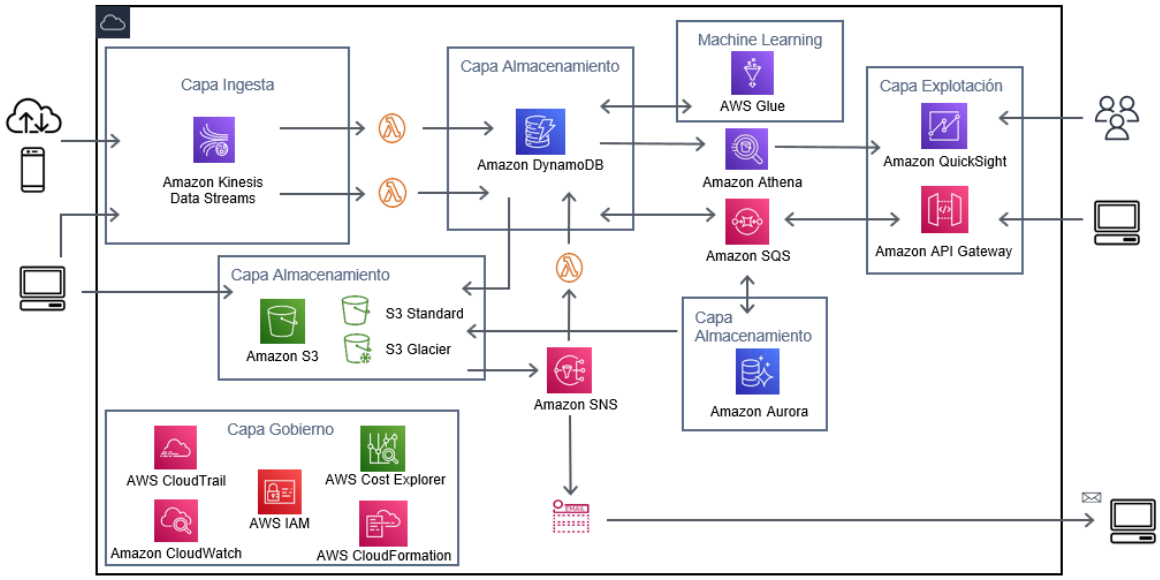

Imagen 7: Arquitectura AWS completa

4.1 Capa Ingesta

La primera capa que vamos a ver es la capa de ingesta. La capa de ingesta es la antesala de la arquitectura. De esta capa depende que todas las demás puedan operar correctamente ya que necesitan de los datos que se extraen en ésta. En esta extracción de datos, como ya hemos comentado anteriormente, disponemos de varias fuentes de datos que podríamos dividir en dos flujos, el flujo online y el flujo batch.

4.1.1 Flujo Online

El flujo online es aquel que recibe los datos de las ventas de cada tienda física, de las ventas del portal web y de la información de las visitas al portal web. Tal y como se ha mencionado en el análisis del problema, contamos con una volumetría diaria de 100.000 registros de las visitas web, 10.000 de las ventas online y 1000 de las tiendas físicas. Todo esto suma un total de 111.000 registros al día.

Para lidiar con esta cantidad hemos decidido hacer dos streams para diferenciar los datos de ventas (tanto online como físicas) de los datos de visitas web. Los registros, al ser ficheros JSON sencillos, únicamente ocupan 0.5KB, y esto, nos acaba suponiendo un total de 50 MB diarios en el stream web y 5,5 MB diarios en el stream de ventas. Podemos apreciar en la Imagen 8 la arquitectura del flujo online y los dos streams por separado.

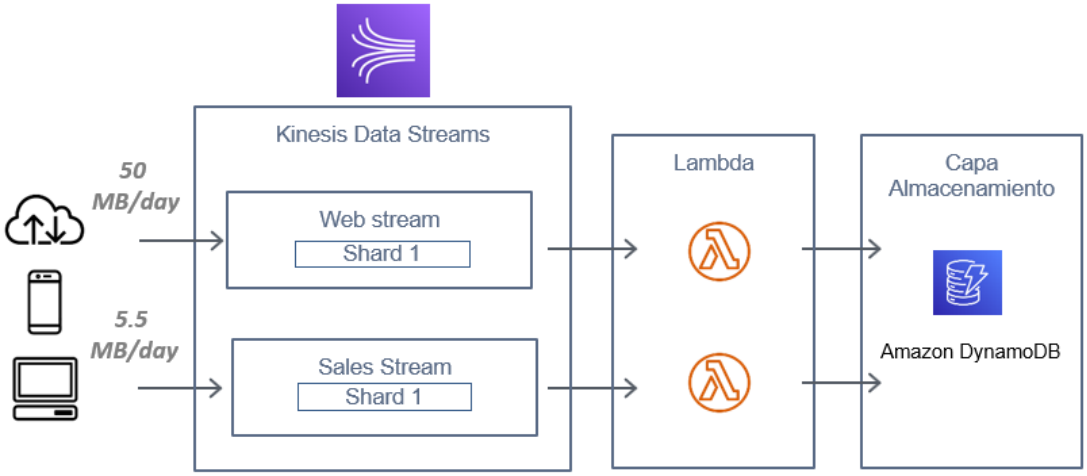

Imagen 8: Arquitectura flujo online

Para crear los streams tendremos que acceder al servicio Kinesis Data Streams en la consola de AWS y crear una secuencia de datos.

Una vez seleccionado esto, elegiremos el modo de capacidad aprovisionado e introduciendo nuestra volumetría en el calculador de particiones nos recomienda el número de particiones óptimo para nuestras necesidades. Al tener una volumetría reducida, lo ideal es utilizar una única partición o *shard* para nuestro stream. Este proceso lo repetiremos de nuevo para crear el otro stream.

Una vez están creados los streams ya se puede configurar el productor de los datos. El productor de los datos será Kinesis Agent [22], una aplicación que, una vez instalada en los equipos locales de las tiendas, monitorizará sus directorios para enviar todos los datos nuevos a Kinesis Data Streams, asegurándonos de esta forma tener siempre un flujo automático de datos.

Para almacenar los datos una vez los hemos extraído, haremos uso de dos funciones Lambda que se encargarán de recoger los registros de Kinesis y los almacenarán en una tabla DynamoDB creada previamente. Configuraremos las dos funciones Lambda para que ejecuten un script en lenguaje Python y les añadiremos un rol de ejecución con permisos para acceder tanto al flujo de Kinesis como a la tabla DynamoDB.

Una vez están configuradas les podremos asignar de desencadenador un stream de Kinesis a cada una. En la Imagen 9 podemos ver el código Python que se utiliza, el cual será igual para las dos funciones Lambdas.

lambda_function

```python
import base64
import boto3

def lambda_handler(event, context):
    dynamodb = boto3.client("dynamodb")
    for record in event["Records"]:
        payload = base64.b64decode(record["kinesis"]["data"]).decode("utf-8")
        dynamodb.put_item(TableName="salesTable",Item={"id":{"S": payload}})
    return "Successfully processed {} records.".format(len(event["Records"]))
```

Imagen 9: Función Lambda Kinesis a DynamoDB

Al principio del código importamos las librerías necesarias para su funcionamiento, que son las siguientes:

- **base64** [23]: Esta librería se utiliza para codificar y descodificar caracteres de diferentes formatos. En nuestro caso la vamos a utilizar para descodificar los registros que recibimos de Kinesis antes de insertarlos en la tabla DynamoDB
- **boto3** [24]: Esta librería se emplea para interactuar con los servicios de AWS. En nuestro caso la utilizaremos para conectarse a un cliente DynamoDB y que pueda insertar los datos en la tabla especificada.

Una vez importadas las librerías, definimos la función Python *lambda_handler* que será la que se encargue de recibir los registros del desencadenador Kinesis mediante el parámetro *event* que recibe. Dentro de la función lo primero que hacemos es asignar a la variable *dynamodb* el cliente de AWS de dynamodb haciéndonos uso de la librería boto3. La librería boto3 permite conectarse a distintos servicios de AWS e interactuar con ellos mediante código Python. Esto nos será de utilidad para acceder o modificar datos de servicios como DynamoDB o S3.

Después recorremos el parámetro *event* con un bucle *for* especificándole que solo recorra los registros de dicho parámetro. En cada iteración del bucle, descodificamos el registro que acabamos de obtener y lo insertamos en la tabla DynamoDB gracias a la función *put_item*. En esta función únicamente le tenemos que especificar el nombre de la tabla a la que se le va a hacer la inserción y la estructura del objeto que se está insertando.

4.1.2 Flujo Batch

El otro flujo sería el flujo batch, que es denominado así ya que únicamente recibe los datos en lotes de forma diaria [25]. En este flujo contamos con una pequeña encuesta que se les adjunta a los clientes después de la compra en la que pueden indicar la valoración de la satisfacción de la compra y el código postal de su vivienda. Estas encuestas se almacenan a lo largo del día en un fichero json y se insertan en el bucket S3 una vez se ha acabado el día o una vez llegue a un mínimo de registros. Para obtener una visión más concreta de esta parte, vemos en la Imagen 10 como sería la arquitectura del flujo batch.

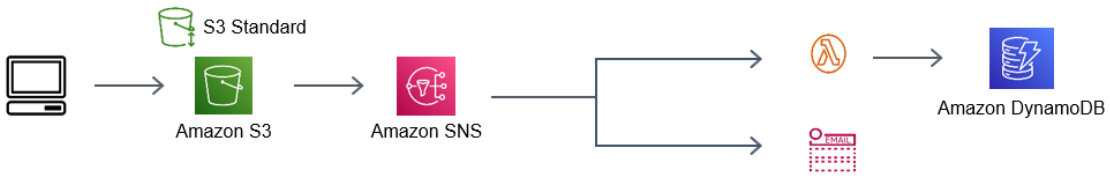

Imagen 10: Arquitectura flujo batch

Los datos se almacenarán inicialmente en un único bucket S3 para todas las tiendas y, posteriormente, mediante una Lambda disparada por el SNS, se introducirán en la tabla DynamoDB. El bucket tendrá bloqueado el acceso público, pero se podrá acceder desde las tiendas a él para la carga de datos gracias a un punto de acceso. El punto de acceso permitirá a los equipos locales de las tiendas tener acceso al bucket S3 para insertar los datos. En la Imagen 11 se aprecia como debemos de configurar el bucket S3 para que gente externa a la empresa no pueda introducir u obtener datos.

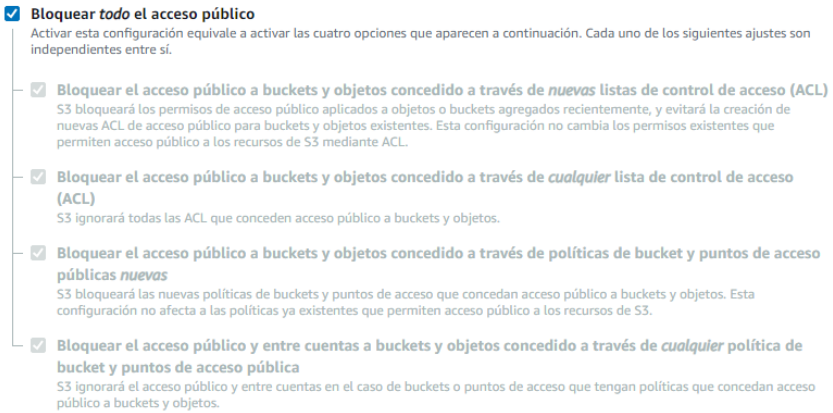

Imagen 11: Configuración de acceso al bucket S3 del flujo batch

Después de la creación del bucket S3 haríamos uso del servicio Amazon SNS. Este servicio nos permite crear un tema para que cuando haya un desencadenador concreto, se encargue de notificar que todo está correcto. La notificación se hace a través de una suscripción donde se le especifica el método a notificar (SMS, Email, etc.).

Una vez tenemos el servicio de notificación funcionando, configuraríamos en el bucket S3 una notificación de eventos para que cuando detecte que hay una inserción de datos en el bucket, lo notifique al tema SNS. En nuestro caso lo hemos configurado para que notifique por correo electrónico a las tiendas una vez la carga de datos se ha realizado correctamente.

Los datos se cargarán desde cada tienda local una vez al día mediante un comando del AWS CLI, que es la terminal de comandos de AWS.

```
aws s3api put-object –-body {data.json} –-key {data/data.json} --bucket
{arn}:{id_cuenta}:accesspoint/{accesspoint-name}:
```

Imagen 12: Comando para insertar los datos del flujo batch en el bucket S3

Este comando (Imagen 12) se ejecutará de manera automática diariamente y hará uso del punto de acceso creado previamente para poder acceder al bucket. Únicamente le tenemos que especificar los siguientes parámetros:

- **body**: En este parámetro tenemos que especificar el archivo que se desea cargar en el bucket S3. En nuestro caso sería un fichero JSON donde se guardaría la información recibida de los clientes acerca de su valoración y su código postal.
- **key**: El parámetro key indica el nombre que se le desea asignar al archivo que se le carga para que así aparezca en el bucket S3.
- **bucket**: En el bucket tenemos que indicar la dirección ARN que obtenemos al crear el punto de acceso, donde se especifica el propio ARN, la id de la cuenta que lo ha creado y el nombre del punto de acceso.

Notificar por email no es la única función que hace el servicio SNS. También nos servirá para que una vez haya recibido notificación del bucket S3, dispare una función Lambda que se encargará de procesar los datos y almacenarlos en una tabla DynamoDB.

Esta función Lambda la configuraremos para ejecutar un script de lenguaje Python y le asignaremos un rol de IAM en el que pueda acceder a los datos del bucket S3 y pueda escribir en la tabla de DynamoDB. Los roles de IAM permiten configurar los permisos asignados a diferentes usuarios según los servicios de AWS a los que deban acceder.

Lo primero que haremos en la función (ver Imagen 13) es importar la librería boto3, ya explicada anteriormente, para conectarnos a los clientes tanto de S3 como de DynamoDB y poder hacer uso de sus funciones. Después también importaremos la

librería json [26], que nos será de utilidad para poder trabajar con ficheros formato JSON, ya que los datos batch se suben al bucket en este formato. Una vez están importadas las librerías, definimos los clientes de S3 y DynamoDB y procedemos a crear la función *lambda_handler*. Lo primero que haremos será obtener el archivo *data.json* del bucket batch. Este archivo es donde se realiza la carga de datos de las encuestas diariamente. Una vez tenemos el archivo, utilizaremos la función *json.loads* para convertir el fichero json en un diccionario Python y que sea más fácil trabajar con él. Una vez esto, se recorre el diccionario y se va insertando cada registro en la tabla DynamoDB "batchData".

lambda_function

```
1  import json
2  import boto3
3
4  s3_client = boto3.client('s3')
5  dynamodb = boto3.client("dynamodb")
6
7  def lambda_handler(event, context):
8      obj = s3_client.get_object(Bucket='batchBucket', Key='data/data.json')["Body"].read().decode("utf-8")
9      file_json = json.loads(obj)
10     for id in file_json:
11         dynamodb.put_item(TableName="batchData",Item=id)
```

Imagen 13: Función Lambda S3 a DynamoDB

4.2 Capa Almacenamiento

La segunda capa que analizar es la capa de almacenamiento. Esta capa es la que se encarga de almacenar los datos extraídos en la capa de ingesta para que posteriormente, los servicios de la capa de procesamiento y explotación puedan hacer uso de ellos en sus tareas y no haya ningún riesgo de pérdida de información. Podemos dividir la capa de almacenamiento en tres partes diferenciadas.

- Las tablas DynamoDB, donde se almacenarán los datos en tiempo real de las ventas, las visitas web y los datos batch.
- Las tablas maestras Aurora MySQL, donde se guardará información que no es tan propensa a los cambios como información de los empleados, productos o tiendas
- El bucket S3 de copia de seguridad. En este bucket se guardará una copia de seguridad de los datos almacenados en las tablas DynamoDB y en las tablas maestras Aurora MySQL.

4.2.1 DynamoDB

Para los datos en tiempo real utilizaremos DynamoDB [27], un servicio de AWS para gestionar bases de datos NoSQL.

Este servicio nos permitirá almacenar la información del stock de los productos y la información proveniente de los dos flujos que hemos visto en la capa de ingesta ya que en el flujo online vimos como mediante Kinesis Data Streams y dos funciones Lambda, podíamos insertar los datos en tiempo real en las tablas DynamoDB. El flujo batch en cambio, pese a pasar anteriormente por un bucket S3, también acaba en su propia tabla DynamoDB.

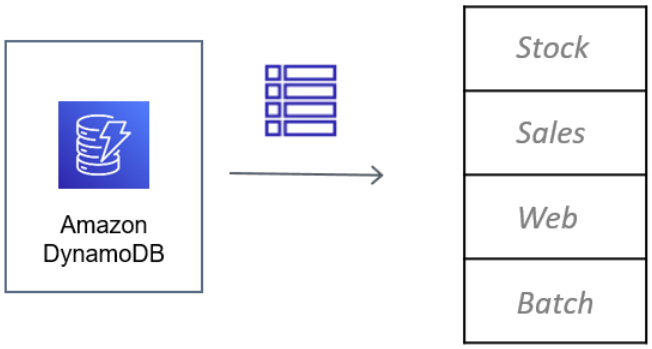

Imagen 14: Tablas DynamoDB

Como podemos observar en la Imagen 14, contaremos con 4 tablas que almacenarán la información comentada anteriormente. Las tablas las podemos crear desde la consola de AWS donde únicamente tenemos que especificar el nombre de la tabla y la clave de partición que es el atributo que hace de identificador de la tabla y permite a DynamoDB recuperar los elementos necesarios en una consulta.

Posteriormente tenemos que elegir la clase de tabla pudiendo ser una tabla estándar o una tabla de acceso poco frecuente, más ideada esta última para datos a los que no se accede asiduamente. Por tanto, elegiremos tablas estándar y el modo de capacidad será on-demand (ver Imagen 15), permitiendo así pagar únicamente por lo que consumimos y poder tener más capacidad de lectura y escritura en caso de que tuviéramos un gran pico de ventas como podría ser en rebajas o con una promoción especial. Una vez hemos creado las tablas, ya podremos almacenar los datos en ellas.

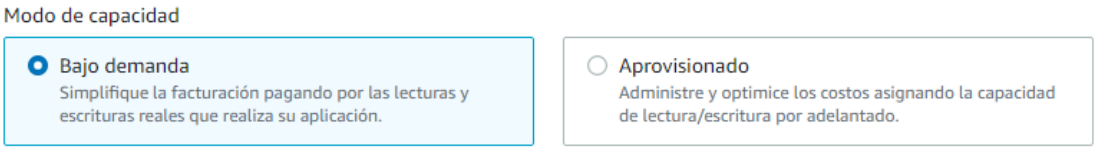

Imagen 15: Modos de capacidad tabla DynamoDB

En esta sección vamos a describir y profundizar en los datos que contendrán dichas tablas. Cada tabla debe contar con una *partition key*, que es una clave primaria única que permite identificar cada registro para facilitar la consulta de datos. Empezando por la tabla stock, podemos ver en la Imagen 16 los datos representados en una tabla y su equivalente en un fichero JSON.

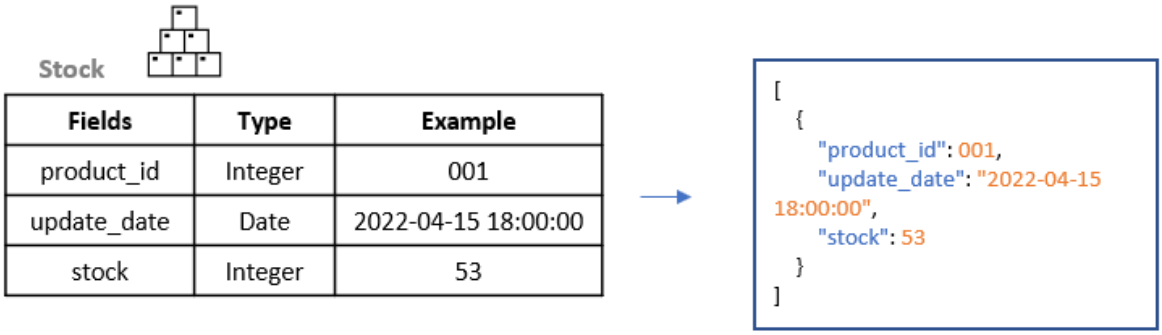

Imagen 16: Ejemplo de tabla y fichero JSON de la tabla de Stock

En la tabla Stock contamos con los siguientes atributos:

- **product_id:** El atributo product_id representa el identificador de un producto en concreto. Se representa con un numero entero y al ser un identificador debe ser único. Este atributo es el que se configura como *partition key* para la tabla Stock

- **update_date:** El atributo update_date representa la última fecha en la que se actualizó el stock de dicho producto. Se representa con un formato de fecha (%Y-%m-%d %H:%M:%S)

- **stock:** El atributo stock representa la cantidad de ese producto en concreto que hay disponible en el momento que se actualizó. Se representa con un número entero.

La segunda tabla es la tabla Sales, donde se almacena información de las ventas que se han realizado. Es probablemente la tabla más compleja ya que dentro de un atributo cuenta con un conjunto de dos atributos más. En la Imagen 17 se aprecia como sería la estructura de dicha tabla y su equivalente en formato JSON.

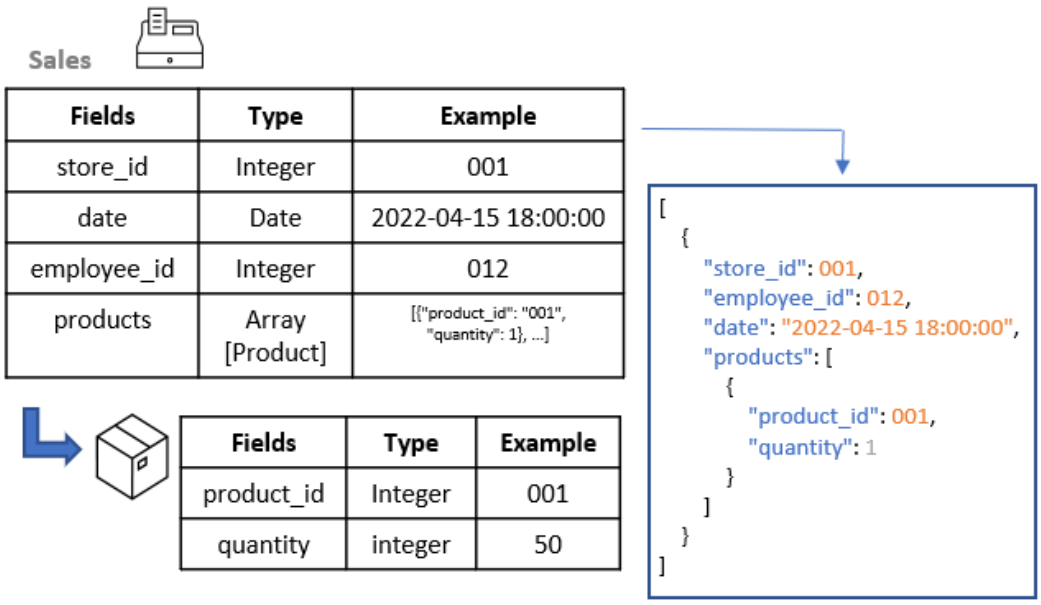

Imagen 17: Ejemplo de tabla y fichero JSON de la tabla de Sales

En la tabla Sales observamos los siguientes atributos:

- **store_id:** El atributo store_id indica el identificador de la tienda donde se realizó la venta. Se representa con un número entero. Este atributo es el que se configura como *partition key* para la tabla Sales.

- **date:** El atributo date indica la fecha en la que se realizó la venta. Se representa con un formato fecha.

- **employee_id:** El atributo employee_id indica el identificador del empleado que realizó la venta. Se representa con un número entero.

- **products:** El atributo products es un conjunto que agrupa dos atributos diferentes. Dentro de este conjunto tenemos el atributo **product_id** que indica el identificador del producto que se ha vendido representado como un número entero y el atributo **quantity** que indica la cantidad del producto indicado en product_id se han vendido en esa transacción. Se representa también con un número entero.

La tercera tabla es la tabla Web. En esta tabla se almacena información acerca de la interacción que tienen los usuarios en la web y cuánto tiempo pasan en ella. Hemos decidido recopilar y analizar esta información en vez de utilizar servicios existentes que ya lo hacen, como Google Analytics, debido a que queríamos tener un mayor control sobre la información que obtuviéramos de los datos. Se puede observar en la Imagen 18 la estructura de la tabla Web y su equivalente en formato JSON.

Web

Fields	Type	Example
session_id	Integer	001
webpage_id	String	"catalogue"
initial_date	Date	2022-04-15 18:00:00
end_date	Date	2022-04-15 18:45:00

Imagen 18: Ejemplo de tabla y fichero JSON de la tabla Web

En la tabla Web distinguimos los siguientes atributos:

- **session_id:** El atributo session_id es el identificador de la sesión web registrada. Está representada con un número entero. Este atributo es el que se configura como *partition key* para la tabla Web

- **webpage_id:** El atributo webpage_id es el identificador de la sección del portal web en la que se encuentra el cliente. Se representa con una cadena de texto.

- **initial_date:** El atributo initial_date es un atributo que nos indica la fecha exacta en la que el cliente se ha conectado a la web. Se representa con un formato fecha.

- **end_date:** El atributo end_date es un atributo que nos indica la fecha exacta en la que el cliente se ha desconectado de la web. Se representa con un formato fecha.

La cuarta y última tabla es la tabla Batch (Imagen 19). Esta tabla es independiente a las tres anteriores ya que no se relaciona con ellas ni cruza datos con las tablas maestras en Aurora MySQL. En ella tenemos los datos que se insertaron previamente en S3 mediante un comando de AWS CLI y posteriormente, se procesaron e insertaron en esta tabla DynamoDB mediante una función Lambda.

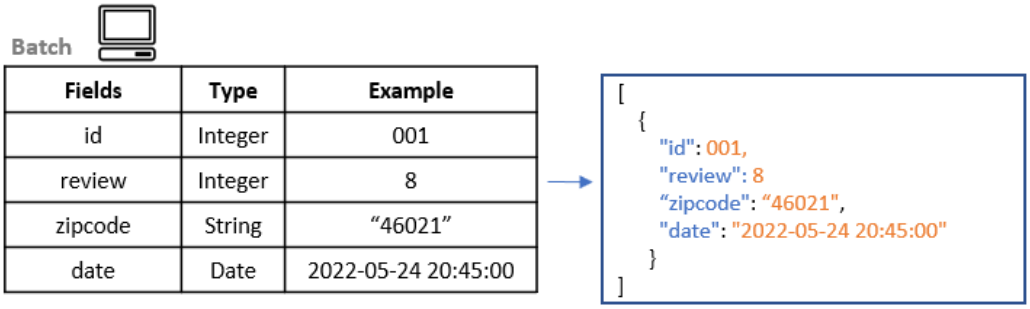

Imagen 19: Ejemplo de tabla y fichero JSON de la tabla Batch

En la tabla Web distinguimos los siguientes atributos:

- **id:** El atributo id nos muestra el identificador de cada encuesta contestada. Se representa con un número entero. Este atributo es el que se configura como *partition key* para la tabla Batch.

- **review:** El atributo review nos muestra el grado de satisfacción del cliente con la transacción realizada en una escala del 1 al 10. Se representa con un número entero.

- **zipcode:** El atributo zipcode nos muestra el código postal del cliente que ha realizado la encuesta. Se representa con una cadena de texto.

- **date:** El atributo date nos muestra la fecha exacta en la que se contestó la encuesta. Se representa con un formato fecha.

Una vez vistas las 4 tablas con las que contamos, vamos a profundizar un poco más en características propias de DynamoDB que pueden sernos útiles. Una herramienta que podríamos utilizar sería configurar las tablas como globales. Esto permitiría reducir la latencia y facilitar la replicación en todo el mundo. Sería algo muy interesante a añadir si observáramos que hay muchas ventas fuera de Europa.

Para tener nuestros datos protegidos y sin riesgo de pérdida, DynamoDB nos da la opción de activar el servicio Point-In-Time-Recovery (PITR). Con este servicio podemos tener copias de seguridad continuas de nuestras tablas de manera automática sin que afecte al rendimiento o a la disponibilidad de las tablas. También podríamos hacer uso de la herramienta Amazon DynamoDB Time to Live (TTL) que nos permite definir una marca de tiempo para determinar cuándo un elemento ya no es necesario. En el caso de que llegue a esa fecha, DynamoDB elimina dicho elemento.

Adicionalmente a estas herramientas anteriores, podremos utilizar el servicio Export to S3 que aprovechando los backups continuos del servicio PITR nos permitirá exportar las tablas deseadas a un bucket S3. Para configurar este servicio únicamente tenemos

que especificar la tabla origen, el nombre del bucket de destino, el formato de los ficheros y la clave de encriptación. Este proceso se puede automatizar para que se realice cada 30 días y así tener cada mes una copia de seguridad adicional en un bucket S3. Como una vez se haya realizado la exportación dichos datos ya no nos serían útiles en nuestra tabla DynamoDB, configuramos el Amazon DynamoDB TTL para que cree una marca de tiempo en la que los objetos se borrarán a los 31 días de estar en la tabla.

4.2.2 Aurora MySQL

Escogeremos el servicio Aurora MySQL para crear las tablas maestras, utilizadas para almacenar los datos más básicos de la base de datos que no son propensos a cambios. En este caso las utilizaremos para que las tablas DynamoDB vistas anteriormente se enlacen con estas y, por ejemplo, teniendo el product_id de la tabla Sales de DynamoDB, podríamos tener la información completa de ese producto en la tabla maestra de Products de Aurora MySQL.

Para crear la base de datos nos permite elegir entre varios motores de bases de datos relacionales como son Amazon Aurora, MySQL, MariaDB, PostgreSQL, Oracle o Microsoft SQL Server. En nuestro caso utilizaremos Amazon Aurora, especificándole que sea una edición compatible con MySQL. También podemos elegir la versión que deseemos del motor de la base de datos, que en nuestro caso será una MySQL 5.7 2.10.2. Después le tenemos que especificar el identificador de la base de datos y las credenciales que se utilizarán para acceder a ella. Posteriormente, la opción más importante a configurar será la clase de instancia que seleccionemos para nuestra base de datos. Depende de los recursos que vayamos a utilizar seleccionaremos una con más capacidad o menos.

Para nuestro caso, al no tener una necesidad de recursos muy alta, seleccionaremos una única instancia t3.small que cuenta con 2 vCPUs, 2 GiB de memoria y 2 GB de almacenamiento. Todo esto será más que suficiente para lidiar con la reducida carga de trabajo que tienen estas tablas maestras. Si la carga de trabajo aumentara, podríamos aprovechar el servicio Aurora Auto Scaling para que desplegara más instancias y no se perdiese rendimiento.

Imagen 20: Tablas Aurora MySQL

Una vez se ha realizado toda la configuración, podremos crear las tres tablas necesarias quedando tal y como se ve en la Imagen 20. Procedemos entonces a profundizar acerca de las tres tablas y los datos que contienen.

En la tabla *Employees* (ver Imagen 21), se almacena la información acerca de los empleados de la empresa. Podemos ver que cuenta con cuatro atributos que son:

- **employee_id:** Este atributo indica el identificador que tiene asignado cada empleado. Se representa con un número entero.
- **name:** Este atributo indica el nombre del empleado en cuestión. Se representa con una cadena de texto.
- **email:** Este atributo indica el correo electrónico del empleado. Se representa con una cadena de texto.
- **store_id:** Este atributo representa el identificador de la tienda donde trabaja ese empleado. Se representa con un número entero.

Employees

Fields	Type	Example
employee_id	Integer	012
name	String	Paco
email	String	paco@email.com
store_id	Integer	001

Imagen 21: Ejemplo de tabla Employees Aurora MySQL

En la Imagen 22 podemos ver la tabla Stores, donde se almacena información acerca de las tiendas físicas. Esta tabla cuenta con 3 atributos, que son:

- **store_id:** El atributo store_id muestra el identificador de la tienda en cuestión. Se representa con un número entero

- **city:** El atributo city muestra la ciudad en la que se localiza la tienda. Se representa con una cadena de texto.

- **email:** El atributo email muestra el email de contacto de la tienda. Se representa con una cadena de texto.

Stores

Fields	Type	Example
store_id	Integer	001
city	String	Valencia
email	String	vlc@email.com

Imagen 22: Ejemplo de tabla Stores Aurora MySQL

La tercera y última tabla es la tabla de Products (Imagen 23). En esta tabla se almacenan datos acerca de la información de los productos de la tienda. La tabla cuenta con 5 atributos, que son:

- **product_id:** El atributo product_id indica el número identificador del producto. Este atributo se representa con un número entero.

- **name:** El atributo name indica el nombre del producto en cuestión. Se representa con una cadena de texto.

- **price:** El atributo price indica el precio del producto. Se representa con un número decimal.

- **registration_date:** El atributo registration_date indica la fecha en la que se registró el producto en la base de datos por primera vez. Se representa con un formato fecha.

- **expiration_date:** El atributo expiration_date indica la fecha en la que el producto se retiró del mercado. Se representa con un formato fecha

Products

Fields	Type	Example
product_id	Integer	001
name	String	Patinete lite
price	Float	199.99
registration_date	Date	2022-01-15 18:00:00
expiration_date	Date	2022-04-15 18:00:00

Imagen 23: Ejemplo de tabla Products Aurora MySQL

Para tener siempre una copia de seguridad de estas tablas que acabamos de ver, Aurora MySQL cuenta con un servicio de snapshots. Este servicio consiste en crear copias de seguridad de las tablas de manera automática cada cierto tiempo. También se le puede especificar el tiempo que se quiere tener almacenadas las copias de seguridad antes de borrarse siendo por defecto un día.

Con el objetivo de añadir una capa de seguridad adicional, exportaremos estas copias de seguridad al bucket S3 donde también se han almacenado las copias de seguridad de las tablas de DynamoDB. Esta exportación se puede realizar también de manera

automática y se le puede especificar si se desea exportar toda la copia de seguridad o únicamente unas tablas concretas.

4.2.3 Amazon S3

Para almacenar las copias de seguridad de todas las tablas mencionadas anteriormente, utilizaremos el servicio Amazon S3. Amazon S3 nos permite crear un bucket, que es un almacén de datos no jerárquico para objetos de todo tipo. Tal y como hemos visto en el apartado del Flujo Batch crearemos también un bucket que tenga bloqueado el acceso público, pero en este caso, no crearemos un punto de acceso ya que no será necesario que las tiendas accedan a éste.

Pese a ser los buckets S3 estructuras no jerárquicas, AWS nos permite implementar una especie de directorios para poder organizar mejor los objetos. En nuestro caso tendremos los datos divididos en dos directorios, siendo uno para los datos de las tablas DynamoDB y otros para los datos de las tablas Aurora MySQL.

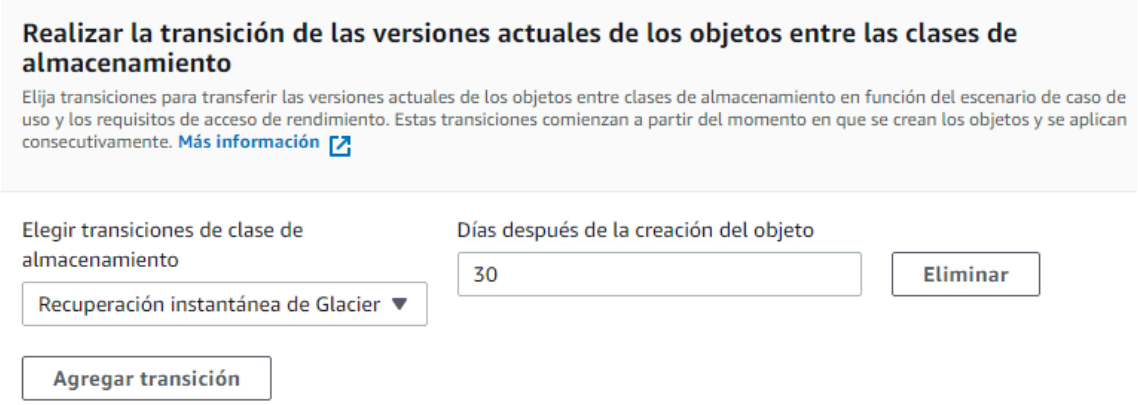

Imagen 24: Configuración ciclo de vida S3

También, como podemos observar en la Imagen 24, configuraremos una regla del ciclo de vida para pasar los datos del S3 Estándar a un bucket S3 Glacier. El ciclo de vida a utilizar consistiría en tener los datos en el bucket S3 Estándar durante 30 días, ya que en ese periodo de tiempo nos serían de utilidad y accederíamos a ellos con frecuencia. Después de esos 30 días, los datos se desplazarían al bucket S3 Glacier, un bucket con un menor coste para que estuvieran 90 días más y entonces se eliminarían (ver Imagen 25).

Esto se debe a que tal y como hemos visto en el análisis del problema, la empresa se ve obligada a almacenar todos los datos durante un mínimo de 120 días, pero solo les darán utilidad a dichos datos durante los primeros 30 días y, por tanto, se desplazarían los siguientes 90 días a un bucket S3 Glacier debido a su reducido coste en comparación a un S3 Estándar

Imagen 25: Ciclo de vida de los datos en S3

4.3 Capa Procesamiento

La tercera capa por explorar es la capa de procesamiento. En esta capa ya tenemos todos los datos extraídos y almacenados y se busca limpiar y transformar dichos datos. Una vez transformados, podremos diseñar un modelo predictivo que permita extraer información acerca de los datos que tenemos. Para todo esto haremos uso del servicio AWS Glue.

4.3.1 AWS Glue

AWS Glue [28] es un servicio de AWS pensado para realizar de una forma sencilla el procesamiento de los datos. Aunque contamos con la opción de utilizar AWS Glue Data Catalog, donde importamos los datos en el propio servicio para agilizar el proceso, en nuestro caso extraeremos los datos directamente desde S3. La herramienta de AWS Glue que gastaremos son los jobs. Los jobs son un servicio de AWS Glue en el que se ejecutan fragmentos de código que se encargan de realizar el procesamiento de los datos. Tal y como observamos en la Imagen 26, hay 5 formas de crear un jobs, que son:

- Crearlo mediante una herramienta visual especificándole origen y destino de los datos
- Crearlo mediante una herramienta visual desde 0.
- Crearlo utilizando un script Spark
- Crearlo utilizando un script de Python
- Crearlo utilizando un Jupyter Notebook

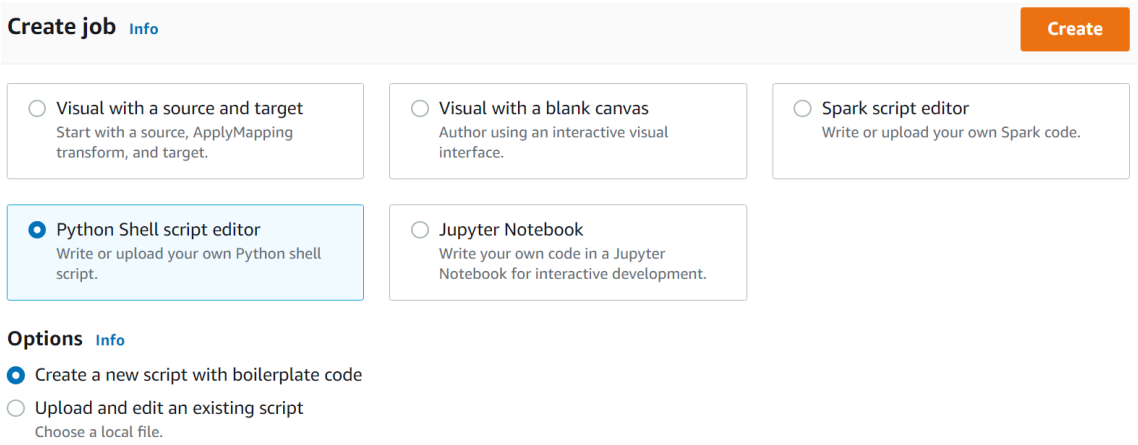

Imagen 26: Tipos de jobs AWS Glue

En nuestro caso, elegiremos crear el trabajo con un script de Python para así poder configurarlo a nuestro gusto. El propio AWS Glue cuenta con editor de código por lo que podemos desarrollar el script ahí sin necesidad de tener que subir el fichero. En la Imagen 27 podemos ver el código que ejecutaría diariamente el job de AWS Glue.

Script Info

```
1  #Importamos las librerías necesarias
2  import pandas as pd
3  import boto3
4  from sklearn.linear_model import LinearRegression
5  from sklearn.model_selection import train_test_split
6  from datetime import datetime
7
8  #Importamos los datos desde el bucket S3 de copia de seguridad
9  s3_client = boto3.client('s3')
10 bucket='backupData'
11 file_key = 'dynamodbExport.json'
12 obj = s3_client.get_object(Bucket=bucket, Key=file_key)["Body"].read().decode("utf-8")
13 df = pd.read_json(obj)
14
15 #Realizamos una regresión lineal
16 X = df["product_id"].values.reshape(-1,1)
17 y = df["stock"]
18 modelo = LinearRegression()
19 modelo.fit(X.reshape(-1,1), y)
20 pred = modelo.predict(X)
21 df["predict"] = pred
22
23 #Exportamos los datos de nuevo al bucket S3
24 now = datetime.now().strftime("%d-%m-%Y %H:%M:%S")
25 file_name = f"dataPredict{now}.json"
26 df.to_json(file_name)
27 s3 = boto3.resource('s3')
28 s3.meta.client.upload_file(file_name, bucket, file_name)
```

Imagen 27: Script AWS Glue job

En la primera parte del código importamos las librerías necesarias para ejecutar el script. Una vez tenemos las librerías, en la segunda parte del código importamos los datos desde el bucket S3 que hace de copia de seguridad. Lo importamos de S3 y no directamente de DynamoDB ya que de esta forma podemos hacerlo directamente y reducir el coste al no depender del servicio AWS Glue Data Catalog [29] . Una vez tenemos el fichero JSON lo transformamos en un dataframe pandas [30] para poder trabajar mejor con él.

La tercera parte del código es la parte del modelo. Hemos diseñado un modelo de regresión lineal que permita analizar y predecir el stock que tendríamos de los productos. Esto lo podremos gastar en un futuro para prevenir posibles ausencias de productos y tener siempre el abastecimiento correcto.

La cuarta y última parte se encarga de exportar los datos obtenidos a un nuevo fichero JSON añadiendo las predicciones respecto al fichero inicial.

Una vez tenemos el código hecho, podemos configurar el job asignándole un nombre y unos permisos mediante AWS IAM. Le tendremos que especificar la capacidad de procesamiento que se utilizará pudiendo elegir entre 1/16 DPU o 1 DPU. Los DPU(Data Processing Unit) son las unidades de procesamiento de datos de AWS Glue. 1 unidad DPU cuenta con 4 vCPUs y 16 GB de memoria por lo que, en nuestro caso, al no necesitar tanta capacidad de procesamiento, escogeremos 1/16 DPU. Además, le podremos asignar el tiempo máximo de espera de la ejecución, que por defecto está en 48 horas y el número de reintentos que hará.

También podemos configurar la ruta del bucket S3 donde se guardará el script, el número máximo de ejecuciones de ese mismo script simultáneamente o añadirle una ruta con librarías Python necesarias para la ejecución del script.

Una vez creado y configurado el job, podremos automatizar su ejecución. Únicamente tenemos que especificar el nombre de la programación del job y la frecuencia. Como queremos se ejecute el script diariamente para tener siempre actualizadas las predicciones, configuraremos la frecuencia como diaria. El propio AWS Glue te da la opción de frecuencia diaria, pero si queremos algo más concreto podemos introducir una expresión CRON [30] como cron(0 10 * * ? *), donde le estamos especificando que se ejecute todos los días las 10 de la mañana.

4.4 Capa Explotación

La cuarta capa es la capa de explotación de los datos. En la capa anterior hemos transformado los datos y hemos obtenido una predicción de cómo serán estos en el futuro. En esta capa nos centraremos en aprovechar toda la información obtenida en la capa anterior y utilizarla para optimizar la toma de decisiones. También la utilizaremos para obtener información en tiempo real de los datos de la empresa. Podemos dividir la capa en dos partes diferenciadas, siendo la primera parte de Amazon QuickSight, más centrada en el análisis y la segunda parte de Amazon API Gateway, más orientada a extraer información en tiempo real.

4.4.1 Amazon QuickSight

Para la parte mencionada anteriormente de análisis utilizaremos el servicio Amazon QuickSight [31]. Amazon QuickSight es un servicio de AWS que permite crear informes donde visualizar los datos de una forma sencilla. Una de sus grandes ventajas es que en un único Dashboard puedes tener información de servicios de AWS como DynamoDB o Redshift, pero también puedes añadir fuentes de datos externas, lo cual lo hace una opción muy completa.

Otra gran característica de Amazon QuickSight es SPICE. SPICE (Super-fast, Parallel, In-memory Calculation Engine) es una herramienta que permite cargar los datos en los dashboards mucho más rápido. Esto se debe a que en vez de necesitar hacer una consulta SQL a la fuente de datos, los datos se almacenan en el propio SPICE reduciendo considerablemente el tiempo de carga.

Para empezar a utilizar el servicio, tenemos que crearnos una cuenta en QuickSight y elegir entre los planes Standard de 12 USD al mes, Enterprise de 24 USD al mes y Enterprise + Q, una edición Enterprise con ventajas añadidas, de 34 USD al mes. En nuestro caso escogeríamos la opción Enterprise ya que nos permitiría tener diferentes lectores del dashboard y que cada gerente de la tienda pudiera acceder a los datos en tiempo real. Una vez elegido esto, ya podremos empezar a confeccionar el dashboard.

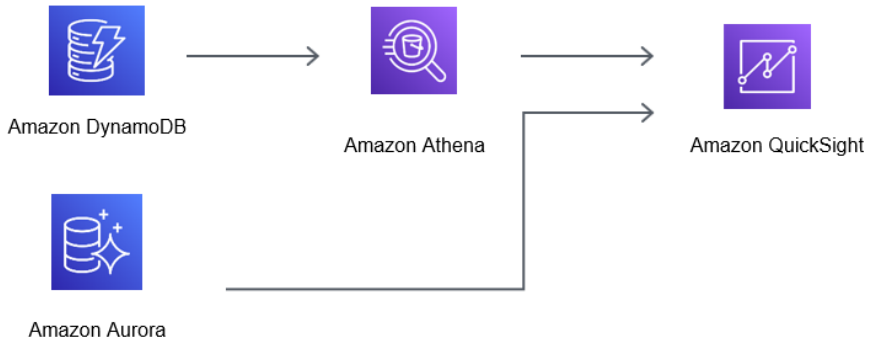

Imagen 28: Fuentes de datos QuickSight

Primero elegiremos las fuentes de datos (ver Imagen 28), que en nuestro caso serán los datos en tiempo real y los batch que tenemos en las tablas DynamoDB y los datos de las tablas Aurora MySQL. Para utilizar los datos de las tablas Aurora MySQL en QuickSight no tendremos que hacer ninguna configuración más ya que al tener integración directa se pueden importar las tablas directamente.

Para importar las tablas de DynamoDB en cambio, al no tener integración directa, tendremos que hacer uso de Amazon Athena. Este servicio se encargará de extraer los datos de DynamoDB e importarlos en tiempo real a QuickSight. Activaremos también el servicio SPICE para almacenar los datos directamente en QuickSight y permitir que haya una carga más rápida de estos. Los primeros 10 GB de almacenamiento de SPICE

están incluidos en la cuota mensual Estándar, por lo que podremos utilizarlos sin problemas al no superar dicha volumetría. Siempre que no excedamos el límite de 10 GB de almacenamiento SPICE, el coste mensual será fijo. Una vez escogido esto, ya podemos confeccionar nuestro dashboard.

Para nuestros datos una opción interesante a utilizar sería una gráfica de barras para poder apreciar la diferencia entre las ventas de cada tienda. También insertaríamos una gráfica de líneas donde se pueda apreciar las ventas totales respecto a las ventas predichas en la capa de procesamiento y una tabla donde repasar los datos de los empleados y ordenarlos por su cantidad de ventas. En la Imagen 29 podemos observar lo que sería un ejemplo completo de un dashboard de Amazon QuickSight con el que representar todos nuestros datos.

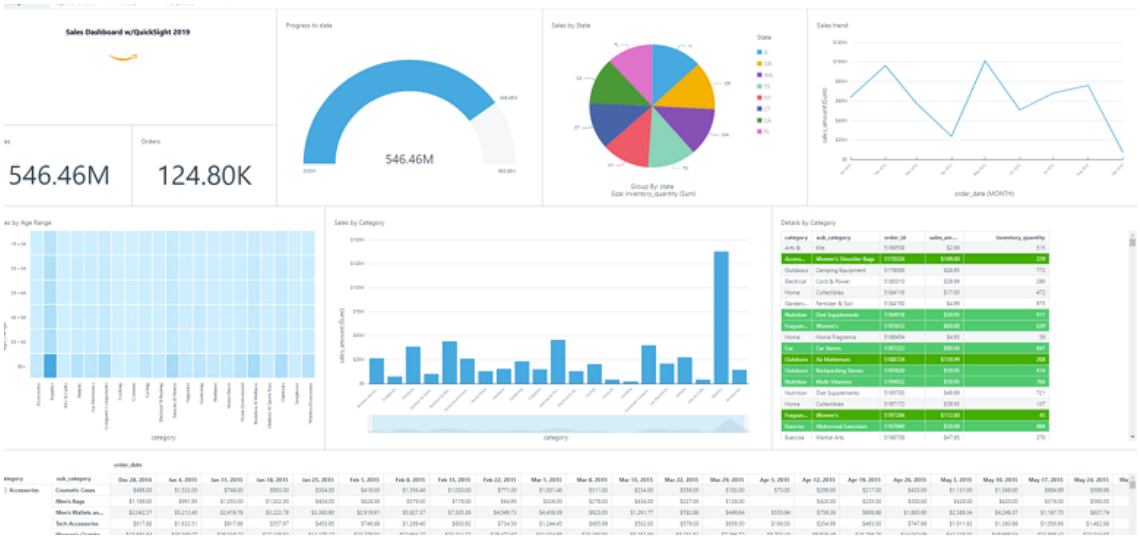

Imagen 29: Ejemplo de dashboard en QuickSight [32]

4.4.2 Amazon API Gateway

Para que en las tiendas físicas se pueda acceder a los datos y modificarlos al instante, haremos uso del servicio Amazon API Gateway [33]. API Gateway es un servicio de AWS que nos permite crear API REST que puedan interactuar con otros servicios AWS. De esta forma, podríamos acceder a los datos de las tablas DynamoDB o Aurora MySQL y realizar llamadas a la API como GET para, por ejemplo, obtener información acerca de un empleado o realizar un llamada POST para introducir un cambio de precio de un producto en su tabla Aurora MySQL. En nuestro caso, utilizaremos dos API REST diferentes, una para los datos de Aurora MySQL y otra para los datos de DynamoDB. Estas APIs podrán ser controladas mediante AWS IAM, permitiendo a los usuarios utilizar sus métodos únicamente si tienen los permisos necesarios.

Para la API REST de Amazon Aurora (ver Imagen 30), crearemos los métodos GET, POST y PATCH. El método GET es el más sencillo de los tres y servirá para extraer información de las tablas directamente, como podría ser información sobre un producto en concreto. Los métodos POST y PATCH son un poco más complejos ya que el método POST inserta nuevos datos y el PATCH inserta modificaciones de los datos ya existentes. Debido a que puede haber riesgo de pérdida de información, hemos decidido implementar dos colas SQS para desacoplar las llamadas a la API y que no se pierda ningún dato. Estas colas tendrán 30 segundos de tiempo de espera, 256 KB de tamaño máximo y un tiempo de retención del mensaje máximo de 4 días. Para conectarse con Amazon Aurora se implementa una función Lambda que hará de intermediaria, permitiendo aplicar una lógica u otra dependiendo de la función de la petición de la API Gateway.

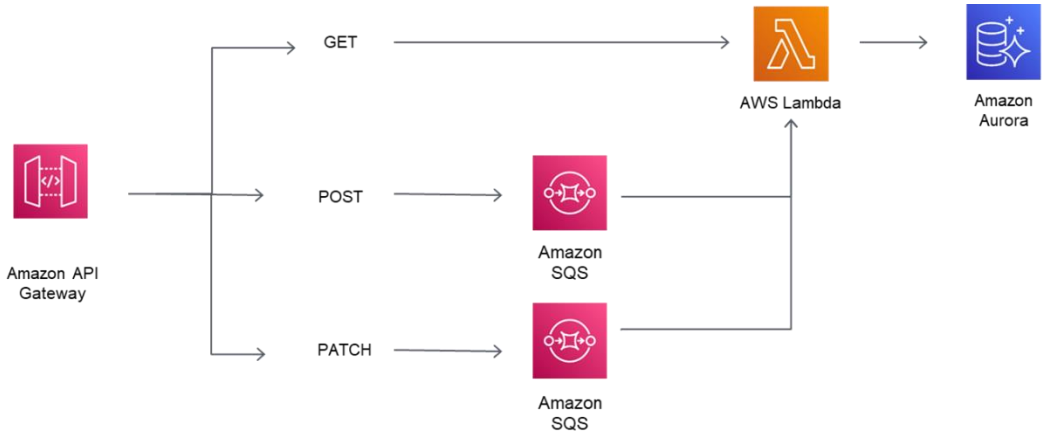

Imagen 30: API REST Aurora MySQL

En la API REST de Amazon DynamoDB (ver Imagen 31), crearemos únicamente los métodos GET y PATCH, ya que el POST en estas tablas no se utilizaría. En esta API se repite el planteamiento de antes y, por tanto, el método GET extraerá la información directamente pero el método PATCH tendrá implementado una cola SQS. Al igual que para la Aurora MySQL, también se implementará una función Lambda para permitir la conexión con Amazon DynamoDB.

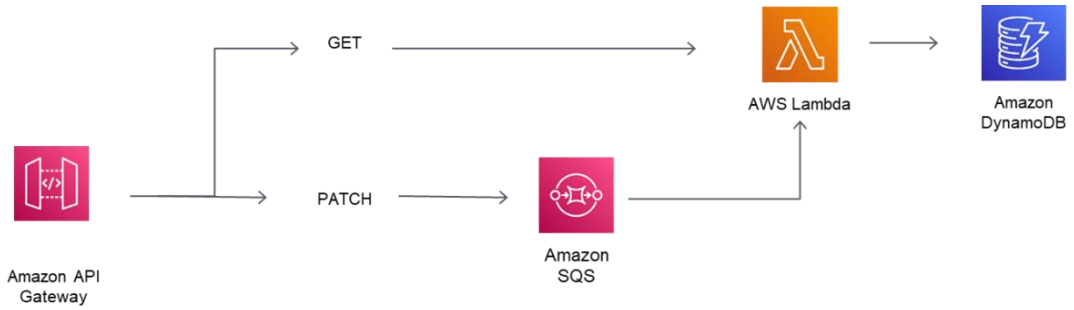

Imagen 31: API REST DynamoDB

4.5 Capa Gobierno

La quinta y última capa es la capa de gobierno. La capa de gobierno [34] es la parte de la arquitectura donde se hace hincapié en la seguridad, usabilidad e incluso la monitorización de los datos. Esta capa la podríamos dividir en 5 partes, siendo cada una un servicio AWS diferente. Las partes, como podemos observar en la Imagen 32 serán AWS IAM, Amazon CloudWatch, AWS CloudTrail, AWS CloudFormation y AWS Cost Explorer.

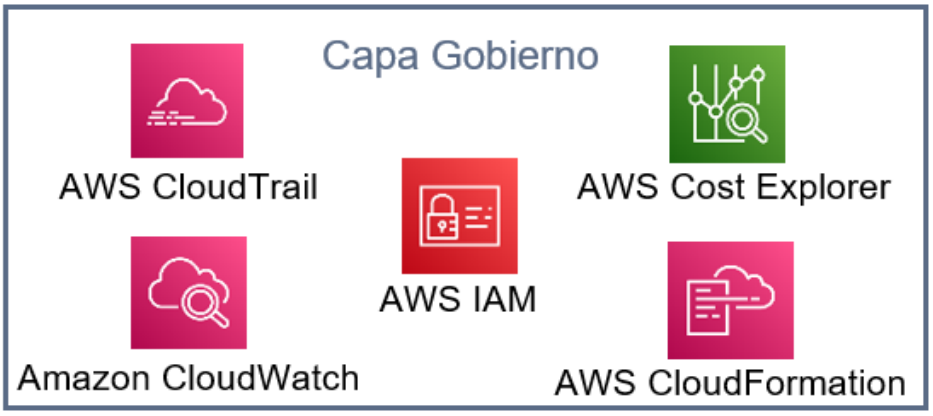

Imagen 32: Servicios de la capa de gobierno

4.5.1 AWS IAM

AWS IAM [35] es un servicio de AWS para controlar y gestionar permisos ya sea para otros servicios o para usuarios de la cuenta. Algunas de sus funciones son permitir que más de una persona administre una misma cuenta AWS o dar acceso a aplicaciones de la cuenta AWS sin tener que compartir información de la cuenta. Otra función muy utilizada es establecer políticas para que los servicios de AWS tengan permiso para interactuar entre ellos. Una gran ventaja de AWS IAM es que todas estas funciones las podemos hacer de manera gratuita.

En nuestro caso, utilizaremos AWS IAM para distintos fines, empezando por crear un usuario para cada tienda física para que tengan los permisos para hacer la carga de los datos batch al bucket S3 provisto para ello. Para configurar esto únicamente tenemos que acceder a la consola de AWS IAM y agregar un usuario nuevo para cada tienda como podemos ver en la Imagen 33. En la configuración le especificamos los permisos que tendrán dichos usuarios, que en este caso serán solamente de acceso al servicio S3.

Nombre de usuario	Grupos	Última actividad	MFA	Antigüedad de la clave activa
tiendaBarcelona	Ninguno	Nunca	Ninguno	Ayer
tiendaMadrid	Ninguno	Nunca	Ninguno	Ayer
tiendaValencia	Ninguno	Nunca	Ninguno	Ayer

Imagen 33: Ejemplo creación usuarios IAM

Una vez tenemos creados los usuarios, se nos proporciona unas credenciales de seguridad que son la clave de acceso y la clave secreta de acceso. Estas credenciales nos servirán para configurar y autentificar el usuario de cada equipo local de las tiendas mediante el comando *aws configure* en AWS CLI tal y como vemos en la Imagen 34, donde se aprecia un ejemplo de las credenciales para la configuración.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

Imagen 34: Ejemplo configuración AWS CLI [36]

También crearemos roles para que las funciones Lambda que hemos configurado tengan los permisos necesarios para su ejecución. Para la Lambda que se encarga de transportar los datos de Kinesis a las tablas DynamoDB hemos creado un rol para que tenga acceso a ambos servicios (ver Imagen 35)

Nombre de la política	Tipo	Descripción
AmazonDynamoDBFullAccess	Administrada por AWS	Provides full access to Amazon DynamoDB via the AWS Management Console.
AmazonKinesisFullAccess	Administrada por AWS	Provides full access to all streams via the AWS Management Console.

Imagen 35: Políticas del rol de la función Lambda de Kinesis a DynamoDB

Para la función Lambda encargada de transportar los datos batch del bucket S3 a la tabla DynamoDB le hemos asignado permisos para que tenga acceso completo tanto al servicio S3 como al servicio DynamoDB (ver Imagen 36).

Nombre de la política	Tipo	Descripción
AmazonS3FullAccess	Administrada por AWS	Provides full access to all buckets via the AWS Management Console.
AmazonDynamoDBFullAccess	Administrada por AWS	Provides full access to Amazon DynamoDB via the AWS Management Console.

Imagen 36: Políticas del rol de la función Lambda de S3 a DynamoDB

A la hora de aplicar roles y políticas a los servicios, debemos seguir en la medida de lo posible el Principio de Privilegio Mínimo. Éste afirma que tenemos que asignar siempre el permiso que más restrictivo sea siempre que sea suficiente para la funcionalidad del servicio. Sin embargo, a modo de ejemplo, se han añadido políticas predefinidas con unos privilegios más elevados.

En el flujo batch hemos desarrollado un sistema de notificaciones mediante SNS para que cuando se introduzcan nuevos datos en el bucket S3, se dispare un evento de notificación hasta SNS. Para que esto pase, necesitamos modificar la política de acceso del tema SNS para que tenga los permisos necesarios para acceder al evento de notificación. Esto se hace mediante un JSON y, como vemos en la Imagen 37, tendríamos que añadir ese código a la política de acceso básica.

```
{
   "Sid": "s3EventSNSNotification",
   "Action": [
     "sns:Publish"
   ],
   "Effect": "Allow",
   "Resource": "<YourTopicARN>",
   "Condition": {
     "ArnLike": {
       "aws:SourceArn": "<Yours3BucketARN>"
     }
   },
   "Principal": "*"
```

Imagen 37: Política de acceso SNS

4.5.2 Amazon CloudWatch

Amazon CloudWatch [37] es un servicio de AWS pensado para monitorizar los recursos que se ejecutan en AWS. Se centra en recolectar métricas y registros de todos los eventos de las aplicaciones y servicios de AWS sin suponer un coste extra ya que es un servicio con una capa gratuita muy extensa y únicamente incurriríamos en gastos cuando quisiéramos monitorizaciones con frecuencias inferiores a 5 minutos. En nuestro caso lo utilizaremos para que, en caso de que las ejecuciones de las funciones Lambda o de los jobs de AWS Glue fallen, podamos tener rápidamente un motivo de el porqué de la ejecución fallida(ver Imagen 38). Los registros se agrupan automáticamente teniendo un grupo de registros para cada servicio que tenga habilitado CloudWatch. Es decir, si tuviéramos 3 funciones Lambda, tendríamos 3 grupos de registros diferentes.

CloudWatch también cuenta con un servicio de alertas que nos permite configurar de antemano para que se dispare cuando sucedan unos eventos especificados. En nuestro proyecto, esta funcionalidad la utilizaremos para que en caso de que el registro de algún

script de AWS Glue tenga un estado de error, esto dispare una notificación mediante el servicio de notificaciones SNS.

Marca temporal	Mensaje
	No hay eventos antiguos en este momento. *Volver a intentar*
2022-06-16T12:34:10.428+02:00	Loading function
2022-06-16T12:34:10.583+02:00	START RequestId: 5a0632ee-69fb-44a4-b5a1-e04c0ab74c3a Version: $LATEST
2022-06-16T12:34:10.597+02:00	[ERROR] KeyError: 's3' Traceback (most recent call last): File "/var/task/lambda_function.py", line 14, in lambda_handler
2022-06-16T12:34:10.599+02:00	END RequestId: 5a0632ee-69fb-44a4-b5a1-e04c0ab74c3a

Imagen 38: Error de Lambda en CloudWatch

4.5.3 AWS CloudTrail

AWS CloudTrail [38] es un servicio de AWS orientado a la auditoría que permite monitorizar la actividad de los usuarios en los diferentes recursos de la cuenta AWS. Se centra en hacer un registro de todas las llamadas a las APIs de los diferentes servicios o recursos.

En nuestro caso, esta herramienta nos podría ser de utilidad para revisar la actividad de los equipos locales de las tiendas físicas en los servicios de AWS. En el panel de CloudTrail, tal y como podemos observar en la Imagen 39 tenemos un historial de eventos en el que rápidamente podemos observar que actividad ha tenido la cuenta en los últimos días.

Historial de eventos Info

Nombre del evento	Hora del evento	Origen del evento
PutScalingPolicy	June 17, 2022, 00:02:55 (UTC+0…	autoscaling.amazonaws.com
UpdateFunctionCode…	June 16, 2022, 15:57:21 (UTC+0…	lambda.amazonaws.com
DeleteAlarms	June 16, 2022, 13:26:48 (UTC+0…	monitoring.amazonaws.com
DeregisterScalableTa…	June 16, 2022, 13:26:48 (UTC+0…	autoscaling.amazonaws.com
AssumeRole	June 16, 2022, 13:26:48 (UTC+0…	sts.amazonaws.com

Imagen 39: Historial de eventos CloudTrail

4.5.4 AWS CloudFormation

AWS CloudFormation [39] es un servicio de AWS que haciéndose uso de plantillas consigue reducir el tiempo invertido en el aprovisionamiento y la configuración de

recursos. Para esto, se utiliza una plantilla de notación de objetos JSON para definir una pila de recursos que funcionan juntos de una forma predeterminada.

En nuestro proyecto sería interesante utilizar una pila donde le definiéramos mediante una plantilla todos los servicios AWS que estamos gastando. Esto permitiría que se crearan, actualizaran o eliminaran todos a la vez si fuera necesario, ahorrando mucho tiempo en el proceso.

Para crear una pila previamente debemos tener la plantilla con los recursos definida o utilizar una plantilla de ejemplo de las que nos ofrece AWS.

Finalmente, este servicio no se utilizó, pero queda abierta la posible implementación en un futuro ya que podría ser de gran utilidad para agilizar el proceso de implementación de toda la arquitectura.

4.5.5 AWS Cost Explorer

AWS Cost Explorer [40] es un servicio de AWS para visualizar y administrar los costes de la cuenta AWS y crear informes personalizados. Dispone de un panel inicial (ver Imagen 40) donde podemos analizar los costos que llevamos en el mes actual y los costes que el propio servicio prevé que tendremos a final de mes. También tenemos un gráfico de barras donde se aprecia el coste desglosado por días.

Esta herramienta es muy útil para nuestro caso ya que permitirá a las tiendas analizar los costes que llevan para que no recurran en sobrecostes inesperados.

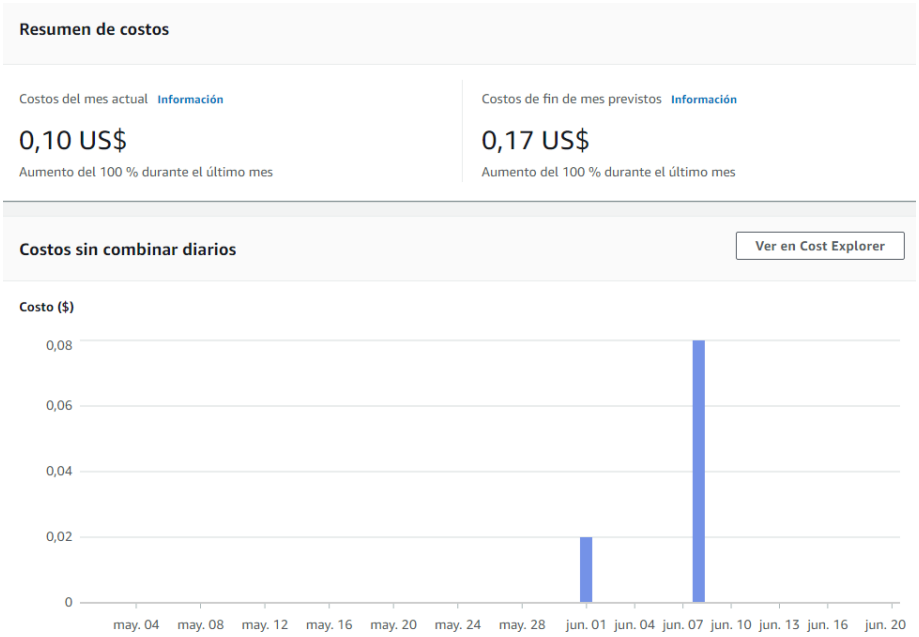

Imagen 40: Panel Cost Explorer

5. Implementación y discusión

Una vez hemos desglosado la arquitectura y explorado todos los servicios que hemos elegido, en este capítulo vamos a aclarar el proceso de implementación de la arquitectura y a analizar otros servicios o herramientas que nos hubieran resultado interesantes de utilizar en nuestro caso de uso y vamos a realizar una comparación de estos servicios con los elegidos finalmente.

5.1 Implementación de la arquitectura

En el capítulo anterior se ha desglosado la arquitectura capa por capa e incluso se ha explicado cómo se realiza la implementación de varios de los servicios utilizados. No obstante, en este proyecto no ha sido posible implementar todos los servicios utilizados en el diseño de la arquitectura debido al coste. Si bien se ha hecho uso, en la medida de lo posible, de la capa gratuita de AWS, hay algunos servicios que no entraban en esta capa gratuita. Amazon Kinesis es un ejemplo de servicio que no entraba en la capa gratuita, pero, al tener un coste muy reducido por hora, se ha podido implementar igual. En cambio, otros servicios como Amazon Aurora o Amazon QuickSight no se han podido implementar debido a su elevado coste, quedándose así únicamente en la fase de diseño.

5.2 Identificación y análisis de soluciones posibles

5.2.1 Kinesis Data Firehose

Kinesis Data Firehose [41] es un servicio para cargar datos en tiempo real. Es similar a Kinesis Data Streams en cuanto a que los dos son parte del servicio Kinesis y están pensados para los datos en streaming.

La gran diferencia que nos ha llevado a utilizar Kinesis Data Streams y no Kinesis Data Firehose es que a pesar de que el propio Firehose cuenta con una herramienta para cargar directamente los datos en diferentes servicios de almacenamiento como son Amazon S3, Amazon Redshift o Elasticsearch, no teníamos disponibilidad de insertarlos en las tablas DynamoDB que teníamos como objetivo, por tanto decidimos hacer uso del Kinesis Data Stream ya que proporciona una menor latencia y mediante una función Lambda sí que podíamos introducir los datos en DynamoDB.

5.2.2 Amazon Redshift

Amazon Redshift [42] es un servicio de almacenamiento de AWS que nos permite tener una base de datos relacional totalmente administrada en la que únicamente tendremos

que elegir la instancia deseada según el rendimiento que queramos. Si bien es cierto que se está desarrollando una versión totalmente serverless de este servicio, aún está en fase de pruebas y por lo tanto no se ha planteado como una opción sólida. Amazon Redshift se haría cargo de los datos en tiempo real que en la arquitectura planteada almacena DynamoDB.

Redshift tiene varias ventajas respecto a DynamoDB, siendo una de ellas su propia estructura. DynamoDB está orientado a bases de datos no relacionales, trabajando con ficheros JSON mientras que Redshift está pensado para bases de datos relacionales, teniendo una estructura mucho más organizada. Otra gran ventaja es el mayor número de integraciones que tiene este servicio respecto a DynamoDB. En caso de haber utilizado Redshift, podríamos tener conectadas directamente la capa de ingesta mediante Kinesis Data Firehose a la capa de almacenamiento de Redshift y a la capa de explotación de Amazon QuickSight, al tener integración directa.

La gran desventaja de este servicio respecto a DynamoDB es su coste. Incluso con la instancia más limitada, la estimación de costes de Amazon Redshift supera con creces la estimación de DynamoDB y al tener un caso de uso donde primaba utilizar servicios más económicos que pudieran hacer el mismo papel, decidimos decantarnos por Amazon DynamoDB.

5.2.3 Amazon RDS

En la arquitectura propuesta hemos hecho uso de Aurora para nuestra base de datos relacional, pero esta no era la única opción disponible. Dentro del servicio de Amazon RDS, había tres opciones que se plantearon como posibles, MySQL, PostgreSQL y Aurora Serverless. La opción ideal habría sido utilizar el servicio Aurora Serverless ya que estábamos planteando una arquitectura serverless y realmente en Aurora MySQL sí que tenemos que hacer una configuración inicial para elegir el tipo de instancia de la base de datos. El mayor inconveniente de Aurora Serverless fue que la versión que había disponible cuando se realizó la arquitectura estaba en fase de pruebas y, por tanto, no estaba recomendada para entornos productivos.

Esto fue el motivo que nos hizo decantarnos por una base de datos Aurora estándar. Dentro de esta, podíamos elegir dos tipos de ediciones, según si iba a ser compatible con MySQL o con PostgreSQL. Acabamos eligiendo una edición compatible con MySQL debido a que, si bien las diferencias de rendimiento no importaban tanto, la instancia más limitada de PostgreSQL era una t3.medium mientras que en MySQL podíamos elegir una t3.small, más adecuada a nuestra volumetría. Esta diferencia suponía el doble del coste mensual en caso de que escogiéramos la opción PostgreSQL.

5.2.4 Amazon S3

Dentro del propio servicio de Amazon S3 hay distintos tipos de buckets a utilizar que se clasifican según la frecuencia y rapidez con la que se busca acceder a los datos [43]. Los principales buckets son:

- **Amazon S3 Estándar:** Es la clase de almacenamiento más utilizada ya que sirve para la mayoría de las situaciones. Es ideal para aquellos casos en los que se accede con frecuencia a los datos y se busca un tiempo de recuperación rápido. No obstante, es el bucket más costoso de todos con un coste de 0,023 USD por cada GB almacenado para los primeros 50 TB al mes. Si superáramos los 50 TB, el coste por cada GB se vería ligeramente reducido. Este bucket fue el elegido para nuestra arquitectura tanto como para almacenar los datos batch como para almacenar las copias de seguridad.

- **Amazon S3 Estándar – IA (Acceso poco frecuente):** Es una clase de almacenamiento muy similar a la primera con la única diferencia de que está pensada para casos en los que se accede con poca frecuencia a los datos, pero cuando se hace se necesita un tiempo de recuperación rápido. La ventaja de esta clase es su reducido coste en comparación a la clase S3 Estándar corriente teniendo un coste de 0,0125 USD por cada GB utilizado.

- **Amazon S3 Glacier:** La clase de almacenamiento S3 Glacier está pensada para tener un almacenamiento de bajo coste al que se va a acceder un número muy reducido de veces cada año. Es ideal para tener siempre una copia de seguridad en caso de desastre natural o perdida de datos. Su coste es de 0,0036 USD por GB almacenado. Este servicio lo utilizaremos mediante el ciclo de vida para pasar los datos que lleven más de 30 días en los buckets S3 Estándar.

- **Amazon S3 Glacier Deep Archive:** La clase de almacenamiento S3 Glacier Deep Archive es la opción más económica de todas. Es un servicio pensado para empresas o administraciones públicas que estén obligadas a almacenar información durante un largo periodo de años. No están ideados para recuperar su información a menudo ya que tienen un tiempo de recuperación de los datos que oscila las 12 horas. Su coste es de 0,00099 USD por cada GB utilizado.

5.2.5 Amazon SageMaker

Amazon SageMaker [44] es un servicio de AWS diseñado para el área de machine learning. Te permite crear, entrenar y desplegar modelos haciendo uso también de datos de otros servicios de AWS. También cuenta con Amazon SageMaker AutoPilot, una herramienta con modelos propios en los que únicamente tienes que especificar la entrada y la salida de datos y no te tienes que hacer cargo del modelado. El propio servicio te hace la fase de ajuste de parámetros para obtener los mejores resultados en

decenas de intentos. La única desventaja de este servicio es su elevado coste respecto a realizar los modelos tú mismo.

Para la capa de procesamiento, Amazon SageMaker era, junto a AWS Glue, las opciones más factibles para utilizar, estando AWS Glue más orientado a trabajos de ETL (Extracción, transformación y carga) y SageMaker a desarrollar modelos más complejos. Uno de los motivos por los que nos decantamos por AWS Glue fue ya no solo por su menor coste si no porque nos permitía automatizar los scripts Python de una forma mucho más simple. Otra razón fue que debido a los escasos datos que teníamos, se desarrolló un modelo de predicción simple, por lo que no necesitábamos todas las herramientas que proporcionaba SageMaker añadiendo complejidad a la capa de procesamiento.

Nuestra solución ideal en Amazon SageMaker hubiera sido utilizar un modelo predictivo XGBoost, el cual tenía una versión integrada en SageMaker facilitándonos así la implementación. Este modelo habría sido desplegado mediante un endpoint, al cual habríamos accedido programando una función Lambda que se ejecutara diariamente e invocara el endpoint del modelo.

5.3 Alternativa al caso de uso

En este proyecto hemos visto el diseño de una arquitectura serverless en AWS donde hemos podido apreciar sus muchas ventajas y observar también su coste. En este apartado se plantea crear una arquitectura alternativa local con herramientas de código abierto a coste 0. La clave de esta arquitectura es la herramienta Docker [45]. Docker nos permite crear contenedores de diferentes servicios y nos da la posibilidad de inicializarlos todos simultáneamente. Es una herramienta muy utilizada también para compartir aplicaciones, ya que con un único comando puedes instalar todo lo necesario con la versión en la que se ideó. Mediante lo que se conoce como *docker-compose* podemos crear un fichero YAML que actuará de plantilla, donde se especificaran los servicios que se van a desplegar. En la Imagen 41 podemos apreciar cómo sería dicha arquitectura.

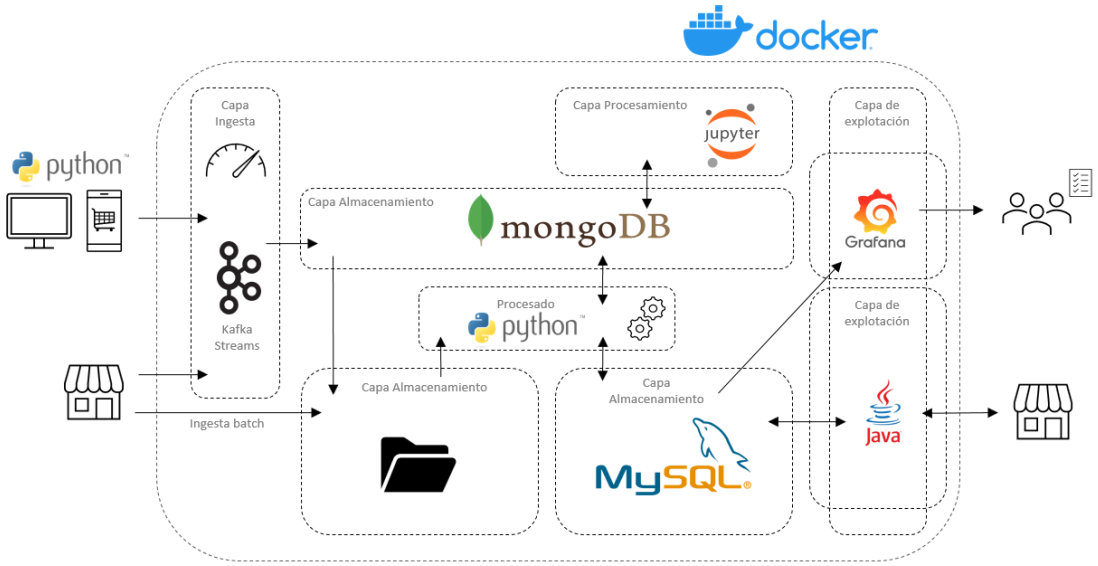

Imagen 41: Arquitectura Open Source

Nos acabamos decantando por una arquitectura AWS y no una local open source debido a la facilidad de integración que teníamos con sus servicios y a la ventaja de no tener que aprovisionar recursos de cómputo ni preocuparnos por su escalabilidad.

En esta arquitectura podemos ver la misma división de capas que en la anterior a excepción de la capa de gobierno.

5.3.1 Capa Ingesta

En la capa de ingesta utilizaremos el servicio Apache Kafka Streams [46] en vez de Amazon Kinesis Data Streams. Este servicio ofrece también transmisión de datos en tiempo real mediante *topics*, lo que en Kinesis conocíamos como streams.

5.3.2 Capa Almacenamiento

En la capa de almacenamiento contamos con 3 diferentes herramientas: una instancia MongoDB [47], una instancia MySQL y el almacenamiento del equipo local. En esta capa, MongoDB es una base de datos no relacional y actúa como lo hace DynamoDB en la arquitectura anterior. Es la que recibe los datos en tiempo real de Apache Kafka y también recibe la ingesta batch del almacenamiento local mediante un script de Python que hace el papel de Lambda. La instancia MySQL almacena las tablas maestras que podíamos ver en la Aurora MySQL en AWS. El almacenamiento local es el sustituto de

Amazon S3, ya que es el que almacena los datos batch y también las copias de seguridad de las tablas.

5.3.3 Capa Procesamiento

La capa de procesamiento está formada por un Jupyter notebook que sustituye el job de AWS Glue. Para realizar el procesamiento, extraería los datos de la MongoDB, los transformaría y los volvería a introducir en la MongoDB. Después, un script de Python desplazaría los datos procesados a una nueva tabla en MySQL para su posterior explotación.

5.3.4 Capa Explotación

La capa de explotación está dividida en dos partes. En la primera contamos con la herramienta Grafana [48], una herramienta open source para la visualización de datos. Al tratarse de una herramienta para realizar dashboards, Grafana sustituiría perfectamente al servicio Amazon QuickSight, del cual su mayor inconveniente es su precio.

Grafana cuenta con integración directa con varias fuentes de datos, siendo una de ellas MySQL. Este el motivo por el que los datos procesados también se han introducido en una tabla MySQL, ya que pese a haber un conector para integrar los datos de una instancia MongoDB en Grafana, dicho conector requiere de la suscripción Grafana Pro. Por lo tanto, al estar buscando una solución con coste 0, no nos será de utilidad.

La otra parte de la capa de la explotación es la API. En la arquitectura de AWS hemos visto como mediante Amazon API Gateway se podía crear fácilmente una API REST con la que insertar u obtener información de las tablas deseadas. En este caso, la API REST la desarrollaríamos en Java, especificándole los mismos métodos que en la API Gateway.

6. Conclusiones y trabajos futuros

En este capítulo de la memoria se va a analizar las conclusiones extraídas de la realización del proyecto y se va a valorar en retrospección si se han cumplido los objetivos establecidos en un principio.

6.1 Conclusiones

Este proyecto se creó con el objetivo principal de diseñar e implementar una arquitectura serverless haciendo uso de la plataforma cloud AWS para solucionar un problema de un comercio particular. Una vez ha finalizado el proyecto, echando la vista atrás se puede afirmar que se ha cumplido el objetivo principal ya que hemos conseguido desarrollar capa por capa todas las partes de una arquitectura completa.

Siempre se ha buscado utilizar los servicios más económicos debido a las necesidades del caso de uso y esto en ocasiones ha generado situaciones más complejas, teniendo que escoger servicios con menos integraciones como es el caso de DynamoDB debido a su reducido coste en comparación a otros servicios.

Personalmente, este proyecto me ha sido de mucha utilidad para formarme en el área de la arquitectura de datos, estudiando alternativas open source como Docker, también en el área del cloud computing y más concretamente con el proveedor AWS. Este proveedor tiene una inmensa cantidad de servicios y te encontrabas constantemente en la situación de tener que decidir qué servicio utilizar entre varios muy similares. Todo esto me ha servido para tener un amplio conocimiento de muchos de los servicios que ofrece AWS. A lo largo del proyecto también he aprendido a documentar exhaustivamente los progresos que hacía y a investigar por mi cuenta posibles soluciones al problema inicial.

Uno de los puntos positivos que le veo al proyecto es haber conseguido reflejar el diseño e implementación de la arquitectura paso a paso. Esto se debe a que, en un futuro, cualquier persona que vea este proyecto podrá observar una solución serverless, que, pese a no ser la solución definitiva, pueda hacer más interesante la computación en la nube animando a intentar los mismos pasos desarrollados en este proyecto.

Este proyecto no hubiera sido posible sin haber cursado antes asignaturas como Bases de Datos o Gestión de Datos para toda la capa de almacenamiento o Infraestructura para el Procesamiento de Datos para toda la arquitectura global. Fue en esta asignatura donde obtuve las nociones básicas de cloud computing que luego profundicé a medida que avanzaba el proyecto. Tampoco hay que restarles importancia a asignaturas como Programación o Estructuras de Datos, ya que sin ellas habría sido mucho más complejo desarrollar los scripts utilizados.

6.2 Trabajos futuros

Pese a haber desarrollado una arquitectura serverless completa, ha habido ciertos puntos que podrían ser interesantes de explorar en un futuro ya que podrían mejorar el proyecto.

Uno de ellos sería ampliar la recogida de datos de las tiendas para tener unas bases de datos más completas con las que tener más opciones para analizar. Una vez tuviéramos unas bases de datos más completas, se podría crear un modelo más complejo que pudiera aprovechar este aumento de datos para realizar mejores predicciones. Para este paso se podría hacer uso de Amazon SageMaker.

Otro trabajo futuro a plantear sería utilizar el servicio CloudFront para mediante una plantilla especificar todos los servicios que se utilizan en la arquitectura y sus configuraciones. Esto permitiría que, si tuviéramos en un futuro un problema similar, podríamos desplegar todos los servicios de una forma mucho más rápida y sencilla.

Una posible mejora a realizar sería mejorar la arquitectura del flujo batch, introduciendo otra API Gateway que mediante una Lambda introduciría los datos en la tabla DynamoDB. Esto nos ahorraría tener que configurar un bucket S3 previo y estar pendientes de la carga de datos diaria y del servicio SNS.

Bibliografía

[1] Smartsheet, «A Complete Guide to the Waterfall Project Method,» [En línea]. Available: https://www.smartsheet.com/content-center/best-practices/project-management/project-management-guide/waterfall-methodology.

[2] Packt, «9 reasons to choose Agile Methodology for Mobile App Development,» 1 October 2018. [En línea]. Available: https://hub.packtpub.com/9-reasons-to-choose-agile-methodology-for-mobile-app-development/.

[3] K. D. Foote, «dataversity.net,» 17 12 2021. [En línea]. Available: https://www.dataversity.net/brief-history-cloud-computing/.

[4] Facultad de Informática de Barcelona, «Historia de Internet,» [En línea]. Available: https://www.fib.upc.edu/retro-informatica/historia/internet.html.

[5] R. Miller, «techcrunch.com,» 2 7 2016. [En línea]. Available: https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/?guccounter=1&guce_referrer=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAAIp1ApXl34bhaHCeiv3hkjkGqrWkcbv6gpSW8L9by5Ysmm24Iz9OHp0rr8jvHBvABvNsCccYEu_oL9XJ8BEJKr8j.

[6] Wikimedia Commons, «Wikimedia,» 2016. [En línea]. Available: https://commons.wikimedia.org/wiki/File:AmazonWebservices_Logo.svg.

[7] S. Paul, «A Cloud Guru,» 12 12 2018. [En línea]. Available: https://acloudguru.com/blog/engineering/history-google-cloud-platform#:~:text=Amazon%20launched%20its%20cloud%20computing,web%20applications%20on%20Google%20infrastructure..

[8] F. Richter, «Statista,» 8 2 2022. [En línea]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

[9] Redhat, «¿Qué es el cloud computing? Diferencias entre IaaS, PaaS y SaaS,» 2 April 2020. [En línea]. Available: https://www.redhat.com/es/topics/cloud-computing/iaas-vs-paas-vs-saas.

[10] Salesforce, «Salesforce,» [En línea]. Available: https://www.salesforce.com/products/platform/best-practices/benefits-of-cloud-computing/.

[11] S. Wickramasinghe, «bmc.com,» 1 10 2021. [En línea]. Available: https://www.bmc.com/blogs/aws-vs-azure-vs-google-cloud-platforms/.

[12] Amazon Web Services, «AWS Pricing Calculator,» [En línea]. Available: https://calculator.aws/#/.

[13] Amazon Web Services, «Amazon Kinesis Pricing,» [En línea]. Available: https://aws.amazon.com/es/kinesis/data-streams/pricing/.

[14] Amazon Web Services, «AWS Lambda Pricing,» [En línea]. Available: https://aws.amazon.com/es/lambda/pricing/.

[15] Amazon Web Services, «Amazon S3 Pricing,» [En línea]. Available: https://aws.amazon.com/es/s3/pricing/.

[16] Amazon Web Services, «Aurora MySQL Pricing,» [En línea]. Available: https://aws.amazon.com/es/rds/aurora/pricing/.

[17] Amazon Web Services, «Amazon DynamoDB Pricing,» [En línea]. Available: https://aws.amazon.com/es/dynamodb/pricing/on-demand/.

[18] Amazon Web Services, «AWS Glue Pricing,» [En línea]. Available: https://aws.amazon.com/es/glue/pricing/.

[19] Amazon Web Services, «Amazon Athena Pricing,» [En línea]. Available: https://aws.amazon.com/es/athena/pricing/.

[20] Amazon Web Services, «Amazon API Gateway Pricing,» [En línea]. Available: https://aws.amazon.com/es/api-gateway/pricing/.

[21] Amazon Web Services, «Amazon QuickSight Pricing,» [En línea]. Available: https://aws.amazon.com/es/quicksight/pricing/.

[22] Amazon Web Services, «Escribir en Amazon Kinesis Data Streams utilizando el agente de Kinesis,» [En línea]. Available: https://docs.aws.amazon.com/es_es/streams/latest/dev/writing-with-agents.html.

[23] Python.org, «Documentación Python,» [En línea]. Available: https://docs.python.org/3/library/base64.html.

[24] Amazon Web Services, «Boto 3 Documentation,» [En línea]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html.

[25] J. Gomar, «Procesamiento batch o por lotes,» 25 11 2018. [En línea]. Available: https://www.profesionalreview.com/2018/11/25/que-es-el-procesamiento-batch/.

[26] Python.org, «Documentación JSON,» [En línea]. Available: https://docs.python.org/3/library/json.html.

[27] Amazon Web Services, «Documentación DynamoDB,» [En línea]. Available: https://docs.aws.amazon.com/es_es/amazondynamodb/latest/developerguide/Introduction.html.

[28] Amazon Web Services, «Documentación AWS Glue,» [En línea]. Available: https://docs.aws.amazon.com/es_es/glue/latest/dg/what-is-glue.html.

[29] Amazon Web Services, «AWS Glue Data Catalog,» [En línea]. Available: https://docs.aws.amazon.com/es_es/glue/latest/dg/populate-data-catalog.html.

[30] Amazon Web Services, «Programación CRON para AWS Glue jobs,» [En línea]. Available: https://docs.aws.amazon.com/glue/latest/dg/monitor-data-warehouse-schedule.html.

[31] Amazon Web Services, «Documentación Amazon QuickSight,» [En línea]. Available: https://docs.aws.amazon.com/quicksight/latest/user/welcome.html.

[32] J. Kunnackal, «Dashboard Amazon QuickSight,» 22 11 2019. [En línea]. Available: https://docs.aws.amazon.com/quicksight/latest/user/welcome.html.

[33] Amazon Web Services, «Documentación API Gateway,» [En línea]. Available: https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/welcome.html.

[34] M. A, «Data Governance o gobernanza de datos: definición y retos,» 14 12 2021. [En línea]. Available: https://datascientest.com/es/data-governance-o-gobernanza-de-datos-definicion-y-retos.

[35] Amazon Web Services, «Documentación AWS IAM,» [En línea]. Available: https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/introduction.html.

[36] Amazon Web Services, «Configuración AWS CLI,» [En línea]. Available: https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html.

[37] Amazon Web Services, «Documentación Amazon CloudWatch,» [En línea]. Available: https://docs.aws.amazon.com/es_es/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.

[38] Amazon Web Services, «Documentación AWS CloudTrail,» [En línea]. Available: https://docs.aws.amazon.com/es_es/awscloudtrail/latest/userguide/cloudtrail-user-guide.html.

[39] Amazon Web Services, «Documentación AWS CloudFormation,» [En línea]. Available: https://docs.aws.amazon.com/es_es/AWSCloudFormation/latest/UserGuide/Welcome.html.

[40] Amazon Web Services, «Documentación AWS Cost Explorer,» [En línea]. Available: https://docs.aws.amazon.com/cost-management/latest/userguide/ce-what-is.html.

[41] Amazon Web Services, «Características Amazon Kinesis Data Firehose,» [En línea]. Available: https://aws.amazon.com/es/kinesis/data-firehose/features/?nc=sn&loc=2.

[42] Amazon Web Services, «Documentación Amazon Redshift,» [En línea]. Available: https://docs.aws.amazon.com/es_es/redshift/latest/mgmt/welcome.html.

[43] Amazon Web Services, «Clases de almacenamiento de Amazon S3.,» [En línea]. Available: https://aws.amazon.com/es/s3/storage-classes/.

[44] Amazon Web Services, «¿Qué es Amazon SageMaker?,» [En línea]. Available: https://aws.amazon.com/es/sagemaker/faqs/.

[45] Docker, «Documentación de Docker,» [En línea]. Available: https://docs.docker.com/get-started/overview/.

[46] Apache Kafka, «Documentación Apache Kafka,» [En línea]. Available: https://kafka.apache.org/documentation/.

[47] MongoDB, «¿Qué es MongoDB?,» [En línea]. Available: https://www.mongodb.com/es/what-is-mongodb.

[48] Shivang, «What is Grafana? Why Use It? Everything You Should Know About It,» [En línea]. Available: https://www.scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/.

[49] Salesforce, «12 Benefits of Cloud Computing and Its Advantages,» [En línea]. Available: https://www.salesforce.com/products/platform/best-practices/benefits-of-cloud-computing/.

[50] S. Viñas, «Así se está convirtiendo España en el 'hub' europeo de los centros de datos,» 24 4 2021. [En línea]. Available: https://www.businessinsider.es/big-tech-invierten-centros-datos-espana-843717.

Anexos

8.1 Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
• **Fin de la pobreza.**				**X**
• **Hambre cero.**				**X**
• **Salud y bienestar.**				**X**
• **Educación de calidad.**				**X**
• **Igualdad de género.**				**X**
• **Agua limpia y saneamiento.**				**X**
• **Energía asequible y no contaminante.**		**X**		
• **Trabajo decente y crecimiento económico.**				**X**
• **Industria, innovación e infraestructuras.**		**X**		
• **Reducción de las desigualdades**.	**X**			
• **Ciudades y comunidades sostenibles**.				**X**
• **Producción y consumo responsables.**				**X**
• **Acción por el clima.**	**X**			
• **Vida submarina.**				**X**
• **Vida de ecosistemas terrestres.**				**X**
• **Paz, justicia e instituciones sólidas.**				**X**
• **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Este proyecto final de grado se puede relacionar con varios Objetivos de Desarrollo Sostenible siendo los más destacables acerca del consumo energético y la dependencia de infraestructura propia para realizar arquitecturas similares.

Una de las mayores ventajas que presentó el cloud computing a la sociedad fue permitir erradicar la dependencia que había hasta entonces de tener siempre tus datos en infraestructuras *on-premise*. Esto permitió que cualquier persona pudiera tener acceso a altas capacidades de computación sin tener que afrontar el gran coste inicial de la instalación de una infraestructura local. Esto por tanto se relaciona de gran manera con el objetivo de reducir las desigualdades, ya que, si bien es cierto que tiene un coste a pagar a los diferentes proveedores cloud, es una fracción de lo que supone todo el gasto y conocimiento técnico de instalarte tu propio datacenter y esto permite que mucha gente se aproveche de estas ventajas para poder desarrollar aplicaciones como la que hemos visto en este proyecto.

Del mismo modo, el ser una tecnología más accesible para todos ha permitido que muchas empresas tanto grandes como pequeñas hayan empezado a migrar a estos servicios debido a la innovación que aporta y al ahorro económico que puede conllevar ya que únicamente se paga por lo que se consume.

En relación con el consumo energético, el cloud computing es una gran manera de ahorrar energía ya que todos los usuarios que la utilizan están ahorrando la energía que gastarían teniendo su infraestructura en local. Esto se debe a que la infraestructura de los proveedores Cloud está organizada en datacenters mucho mejor optimizados ya que aprovechan el mismo equipamiento para dar soporte a diferentes usuarios ahorrando así consumo energético y reduciendo por tanto las emisiones convirtiéndose en una opción mucho más sostenible.

Otro punto para tener en cuenta relacionado con la acción por el clima es no solo que se reduzca el consumo energético como ya hemos visto, si no que los proveedores cloud están cada vez más apostando por energía renovable. Concretamente el proveedor que se utiliza en este proyecto, Amazon Web Services (AWS), se ha fijado un objetivo para 2025 de alimentar todas sus operaciones con energía 100% renovable.

8.2 Glosario

API: Acrónimo en inglés de Application Programming Interface, una API es un mecanismo que permite a diferentes aplicaciones comunicarse entre sí.

AWS: Acrónimo en inglés de Amazon Web Services. Es el proveedor de cloud computing de Amazon y actualmente el más utilizado en el mercado.

Azure: Azure es el proveedor de cloud computing de Microsoft.

Batch: El procesamiento batch o, por lotes en español, indica un tipo de ejecución donde no se insertan los datos por cada registro si no que se insertan en lotes para ahorrar ejecuciones.

Bucket: Un bucket es la figura que utiliza AWS de contenedor de objetos. Está pensado para almacenar cualquier tipo de fichero y no tiene una estructura jerárquica.

Cloud computing: El cloud computing, conocido también como computación en la nube en español, es el suministro de servicios de software como puede ser almacenamiento, procesamiento o virtualización por parte de los proveedores cloud para no necesitar una infraestructura local.

Dashboard: Un dashboard, o cuadro de mando en español, es una herramienta para monitorizar y visualizar datos de una manera sencilla.

Datacenter: Un datacenter o centro de procesamiento de datos es una sala o edificio donde se alojan los servidores de las empresas.

Endpoint: Un endpoint es una URL de una API que responde las peticiones que recibe.

GCP: Acrónimo de Google Cloud Platform, es el proveedor de cloud computing de Google.

JSON: Acrónimo de Javascript Object Notation, JSON es un formato de estructuración de datos.

On-demand: Un servicio on-demand, o bajo demanda en español, es un servicio en el cual se paga por lo que se consume sin tener que hacer frente a un coste inicial ni tener compromisos de pago a largo plazo.

Serverless: La computación serverless, o sin servidor en español, es un modelo de computación en la nube en el que el proveedor cloud es el que administra todos los recursos necesarios, aliviando al usuario de realizar toda la configuración.

SQL: Acrónimo de Structured Query Language, SQL es un lenguaje de programación diseñado para la administración de bases de datos relacionales.