Wo2W: World of Oriented Words

Eduardo López Fortes y Alejandro Jesús González Santana

January 17, 2025

1 Introducción

World of Oriented Words (Wo2W) es una aplicación distribuida y desplegada en la nube de AWS que permite a los usuarios descargar libros del repositorio de gutenber project, indexarlos en una base de datos Neo4j y realizar búsqueda relacionadas con las libros indexados.

El nombre del proyecto, más concretamente el acrónimo Wo2W, hace referencia a la idea de AWS para nombrar sus servicios, que consiste en tomar las primeras letras de cada palabra que conforma el nombre del servicio y unirlas con el número de letras que hay entre ellas: EC2 (Elastic Compute Cloud), S3 (Simple Storage Service).

El proyecto sigue una arquitectura basada en microservicios, donde cada uno de ellos se encarga de una tarea específica. Dichos microservicios se relacionan entre sí a través de los distintos mecanismos de comunicación que ofrece AWS, como por ejemplo, tópicos.

El proyecto se compone de los siguientes microservicios: Crawling Service, Text Processing Service, Datamart Updater Service, Datamart Reader Service, Data Processing Service, Query Service, Dashboard Service y Request Analysis Service.

La Figura 1, la cual representa el diagrama de contexto de la arquitectura, muestra como la aplicación se subdivide en los distintos microservicios y a su vez se diferencian dos grupos de microservicios completamente separados: por un lado se encuentran los microservicios que alimentan el datamart y por otro lado se encuentran los microservicios que se encargan de la interacción con el usuario y la lectura de los datos del datamart.

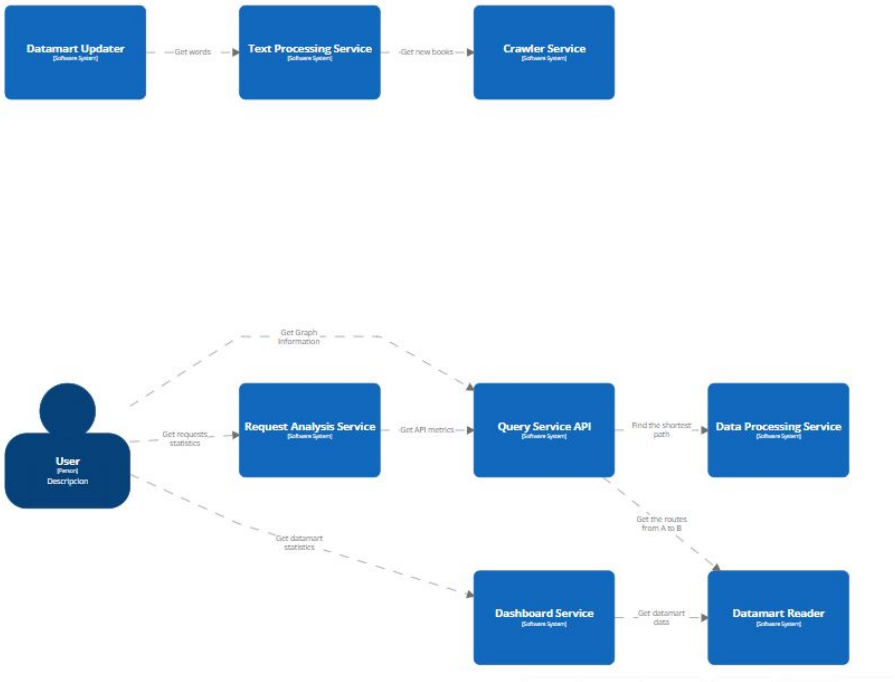

Figure 1: Context Diagram

A lo largo de este documento se explicará brevemente la función de cada uno de los microservicios, así como los componentes usados en su desarrollo. Además, se explicará cómo se ha desplegado la aplicación en la nube de AWS y cómo se ha configurado la infraestructura para que los microservicios se comuniquen entre sí.

2 Crawler or Book Downloader

El Crawler realiza la función de descargar libros en batch desde la página web de project Gutenberg. La función recibe un tamaño de batch a descargar, el id del libro donde comienza la descarga y el id del libro donde finaliza la descarga. La clase Controller dentro de este módulo se encarga de organizar la descarga de libros llamando al Storage correspondiente. En este caso, hemos recurrido a la solución de un s3 de aws donde los libros se almacenan como objetos. Para interactuar con este programa, hay que hacerlo a través de una api programada en Python con flask, que acepta estos atributos como parte de un solicitud HTTP post.

Figure 2: Crawler Diagram

En la figura 2 podemos ver la arquitectura de este módulo, tenemos una api endpoint que en realidad no existe. El crawler se usa como punto de entrada que transmite las solicitudes de descarga las apis en en ejecución en las intancias EC2. Sin embargo, en un principio la idea era emplear una api gateway pero por restricciones del laboratorio (Cloud Foundations sandbox) no pudimos. Cada instancia ec2 se ubica en su propia subnet dentro de una vpc propia para este módulo, que pese a ser pública, contiene un grupo de seguridad que limita el tráfico de entrada que reciba a través del balanceador de carga (ELB). Cada subnet esta en una zona disponibilidad completamente distinta, requisito exigido por ELB. Se probó a crear la subnets de forma privada sin embargo era imposible el tráfico desde elb tras varios intentos. El bucket de s3 donde se almacenan los libros es 's3-wo2w-books'.

3 Text Processing

La Figura 3 servicio de Text Processing se ha implementado como una lambda que se invoca cada vez que un libro se almacena en el datalake, es decir, en un bucket de S3. La lambda se encarga de procesar el texto del libro, i.e., eliminar aquellas palabras cuya longitud sea inferior a 3 o mayor que 5, eliminar caracteres especiales y convertir el texto a minúsculas. Una vez limpiado el texto, se tokeniza de la siguiente manera: para cada palabra se cuenta su número de apariciones en el texto. Una vez tokenizado el texto, se indexa en un diccionario

que se almacena en un bucket de S3. Dicho diccionario contiene una lista de archivos, donde cada archivo corresponde a una palabra y contiene una lista del número acumulado de apariciones de dicha palabra para todos los libros indexados. Una vez indexados, el listado de palabras cuyos archivos de apariciones se han actualizado se envía a un tópico de SNS para que el servicio de Datamart Updater pueda actualizar la base de datos Neo4j de una manera eficiente.

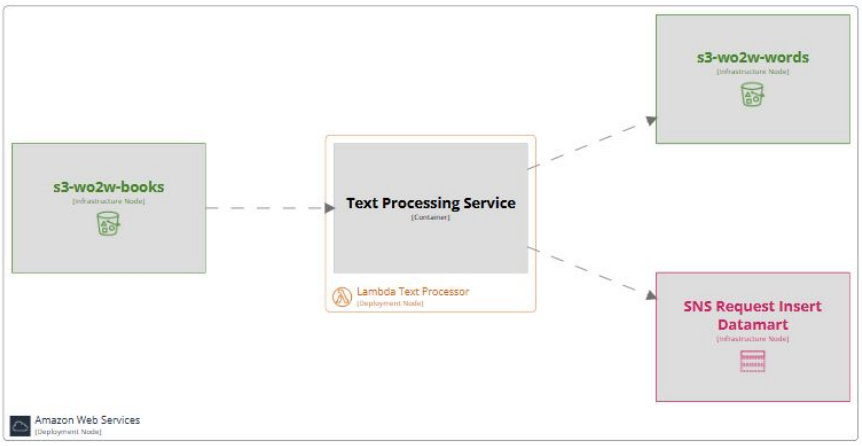

Figure 3: Text Processing Service Diagram

4 Datamart Updater

El servicio de Datamart Updater fue desarrollado en Java y se encarga de recibir una serie de palabras que luego debe leer del bucket de s3 donde se almacenan sus apariciones. Para cada palabra que recibe inserta o actualiza un nodo de la base de datos en grafo, que hemos implementado usando neo4j Aura (en la nube).

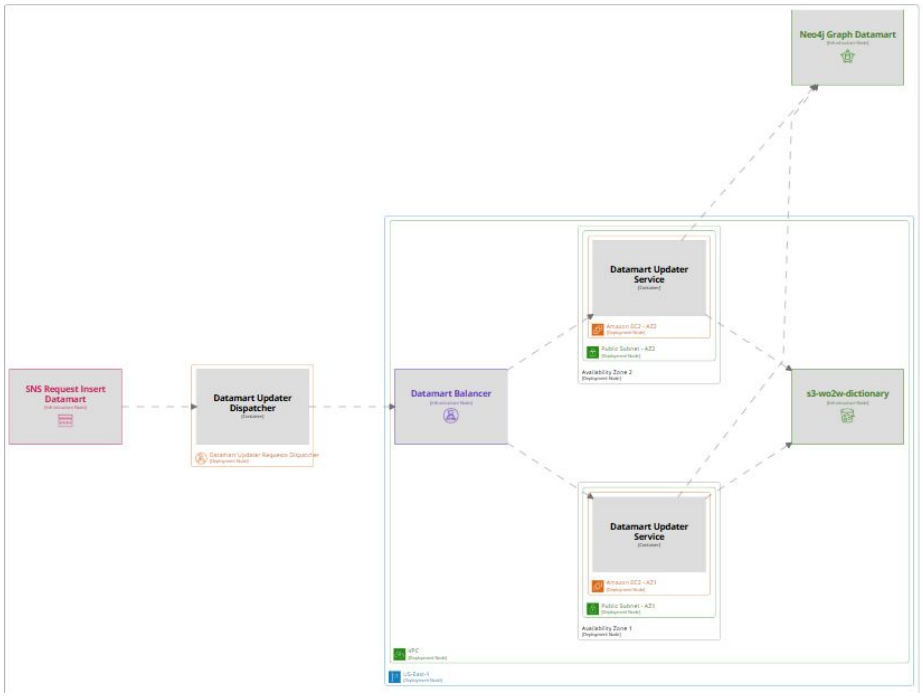

Figure 4: Datamart Updater Diagram

Como vemos en la figura 4, el Datamart updater tiene como punto de entrada una Lambda que se encarga de transmitir al Elastic Load Balancer una solicitud de post en el método upload con una serie de palabras separadas por coma. Elastic load balancer reparte la carga de trabajo en dos ec2 en una estructura prácticamente idéntica a la vista para el módulo de Book Downloader (dos instancias ec2 cada una dentro de su propia subnet en una vpc). En esta ocasión cada ec2 se conecta al bucket s3 de palabras (de la que leen las ocurrencias de las palabras) y a una base de datos de Neo4j que hemos representado con el símbolo de la base de datos de Neptune por falta de icono para Neo4j.

5 Datamart Reader

El servicio de Datamart Reader se encarga de recibir distintas consultas a la base de datos de neo4j, que es el encargado de ejecutar. Las consultas se explicarán en apartado posteriores, donde se explicará como interactuar con las apis correspondientes. El programa de consultas fue realizado en Java y para interactuar con el se dispone de una api. Una vez se realice una consulta, esta será almacenada en un bucket de s3 que se usará como caché y permitirá leer los resultados de forma sencilla en los servicios correspondientes.

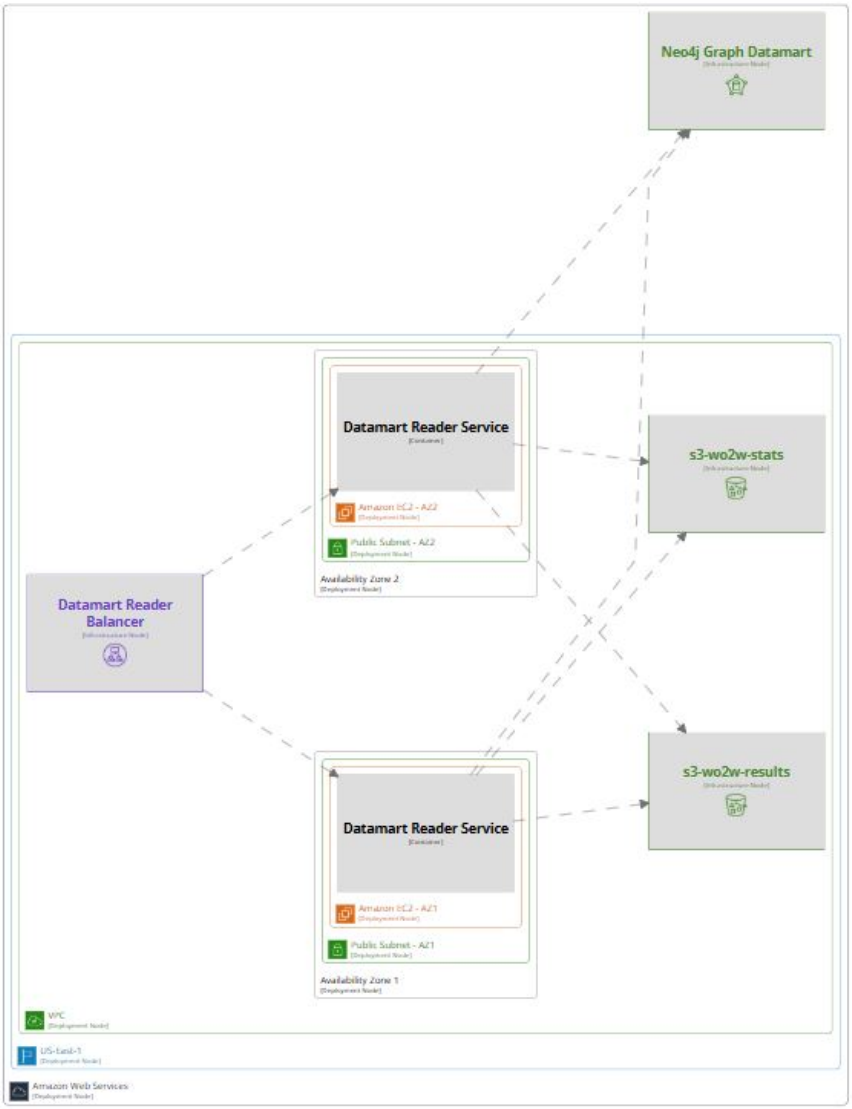

Figure 5: Datamart Reader Diagram

La Figura 5 muestra la arquitectura de este servicio. Como vemos al igual que muchos de los servicios vistos, se basa en un punto de entrada que es un elastic load balancer que se comunica con dos instancias de EC2 para transmitir las peticiones. Estas instancias almacenan los resultados en dos buckets dependiendo de la naturaleza de la consulta. Si la consulta está relacionada con estadísticas del grafo el resultado se guardará en 's3-wo2w-stats', si está

relacionada con palabras y caminos los resultados se almacenarán en 's3-wo2w-results'. Ambas instancias obtienen los resultados leyendo de nuestra base de datos en rgafo alojada en Neo4j.

6 Query Service

El Query Service es la interfaz de usuario de la aplicación. Se encarga de recbir las peticiones del usuario y de enviarle la petición correcta al servicio Datamart Reader. Tal y como se aprecia en la Figura 6, este miceroservicio está implementado como una EC2 en la que se ejecuta un servidor Flask. El servidor recibe las peticiones del usuario y las envía al servicio Datamart Reader a través de otra API, este caso, del Datamart Reader. Una vez que el servicio Datamart Reader ha procesado la petición, el Query Service recoge el resultado y lo envía al usuario. Las distintas peticiones implementadas en la aplicación se describen a continuación:

- findPaths: encuentra el camino más corto entre dos palabras.
- findNodesWithConnectivityDegree: encuentra los nodos que están conectados a un nodo dado con un grado de conectividad igual o mayor al especificado.
- findMaxDistancePaths: encuentra el camino más largo en el grafo.
- findNodesWithHighestConnections: encuentra los nodos con mayor número de conexiones.
- findIsolatedNodes: encuentra los nodos aislados.
- findNodesCluster: encuentra los nodos que forman un cluster.

Para descargar al microservicio Datamart Reader, hemos creado el microservicio Data Processing Service, que procesa el resultado devuelto por el Datamart Reader para las peticiónes que requieren procesamiento adicional.

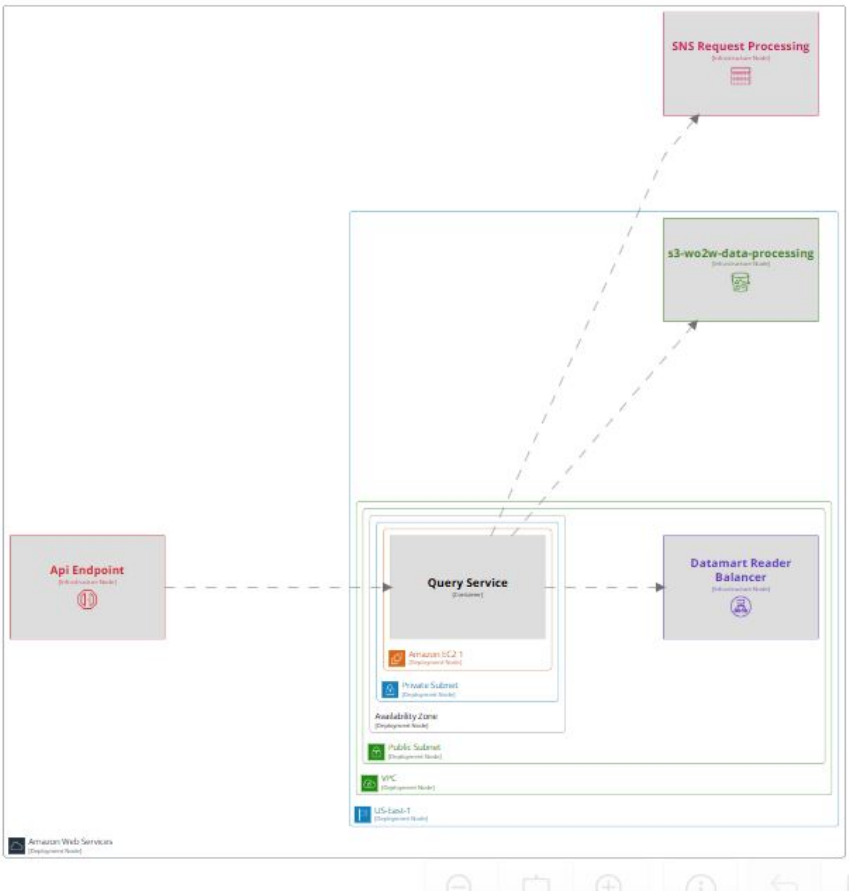

Figure 6: Query Service Diagram

7 Data Processing Service

El Data Processing Service se encarga de procesar el resultado devuelto por el Datamart Reader para las peticiones que requieren procesamiento adicional. Estas son, findPaths y getNodesCluster. Para findPaths, el Data Processing Service se encarga de devolver el camino más corto entre dos palabras. Para getNodesCluster, el Data Processing Service se encarga de devolver los distintos clústeres de nodos que forman el grafo. Para invocar al Data Processing Service, el Query Service envía un mensaje a un tópico de SNS, indicando la petición y el id de la misma. Este mensaje invoca a la lambda (Ver Figura 7) del Data Processing Service, que se encarga de procesar la petición y postetiormente deposita el resultado en forma de archivo en un bucket de S3, cuyo nombre es el id de la petición. El Query Service entonces, espera asta que se haya procesado

la petición, recoge el resultado del bucket de S3 y retorna el resultado al usuario.

Figure 7: Data Processing Service Diagram

8 Request Analysis Service

Este servicio se encarga de almacenar y representar las distintas métricas de uso de la aplicación. Para ello, se ha implementado hecho uso de Prometheus. Prometheus es un sistema de monitorización y alerta de código abierto. Se encarga de recoger métricas del Query Service y almacenarlas en una base de datos de series temporales. Además, Prometheus ofrece una interfaz gráfica para visualizar las métricas almacenadas. Para recoger las métricas de los distintos microservicios, se ha hecho uso de un agente de Prometheus que se ejecuta en el Query Service. Periódicamente, el servidor de Promehteus scrapea las métricas del Query Service y las almacena en la base de datos de series temporales. Las métricas almacenadas son las siguiente:

- Contador de peticiones para cada tipo de petición.
- Contador de máximas peticiones concurrentes.
- Histograma de tiempo de respuesta.
- Resumen de tiempo de respuesta.

Figure 8: Request Analysis Service Diagram

9 Dashboard Service

El Dashboard Service es un microservicio que se encarga de responder a las peticiones del usuario en relación a estadísticas propias del datamart. Este microservicio se ha implementado como una EC2 (Figura 9 en la que se ejecuta un servidor Flask. El servidor recibe las peticiones del usuario y las envía a la API del Datamart Reader. Una vez que el servicio Datamart Reader ha procesado la petición, el Dashboard Service recoge el resultado y lo envía al usuario. Dichas estadísticas son las siguientes:

- findTotalNodesCount: devuelve el número total de nodos en el grafo.
- findTotalConnectionsCount: devuelve el número total de conexiones en el grafo.
- findAverageNodeDegreeCount: devuelve el número medio de conexiones por nodo.
- findIsolatedNodesCount: devuelve el número de nodos aislados.
- findConnectedNodesCount: devuelve el número de nodos conectados.

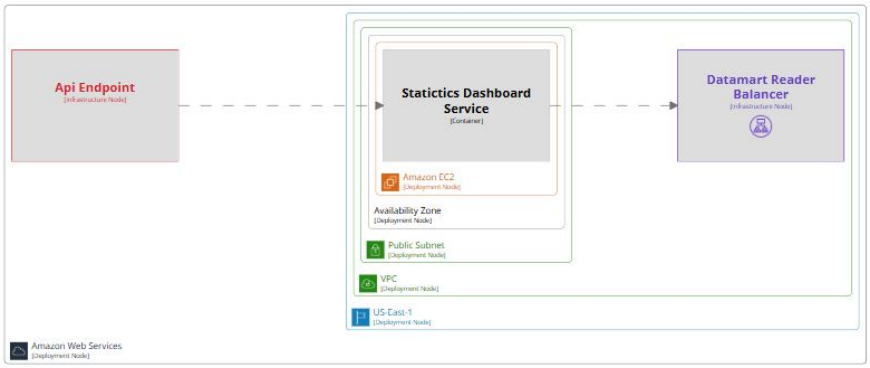

Figure 9: Dashboard Service Diagram