Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Sistemas Computacionales

Análisis de grafos como servicio

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
MAESTRO EN SISTEMAS COMPUTACIONALES

Presenta: **PEDRO ROMERO VARGAS**

Asesor **DR. LUIS FERNANDO GUTIÉRREZ PRECIADO**

Tlaquepaque, Jalisco. enero de 2020.

AGRADECIMIENTOS

Quiero agradecer a mi Familia y amigos que me apoyaron durante el transcurso de esta aventura.

Además, quiero agradecer a Continental, y mis lideres dentro de esa empresa quienes siempre me apoyaron con permisos para completar trabajos a tiempo y permitirme adquirir nuevos conocimientos relevantes para mi crecimiento profesional.

De la misma manera agradezco a mi compañera de posgrado Cyntia por compartirme su conocimiento, punto de vista y brindarme su ayuda cuando la necesité.

Agradezco a mis profesores, quienes facilitaron mi aprendizaje y compartieron su experiencia conmigo, en especial a mi asesor Luis Fernando Gutiérrez Preciado por su conocimiento, asesorías y paciencia durante el desarrollo de este trabajo de obtención de grado.

Al mismo tiempo agradezco a CONACYT por otórgame la beca nacional con número 487148. Y al ITESO por brindarme un lugar apropiado para adquirir conocimientos y el descuento empresarial el cual facilitó mi estudio.

DEDICATORIA

Dedico este trabajo de obtención de grado a mi familia y amigos.

RESUMEN

El análisis de grafos es una tarea compleja pero que es de mucha utilidad tanto para la academia como la industria debido a que los resultados de estos análisis pueden generar gran valor en el área de interés donde se utilicen, además estos pueden ser utilizados en múltiples áreas de la ciencia.

Es por lo que hoy en día existen múltiples herramientas y librerías que nos ayudan con esta tarea, sin embargo, muchas veces nos encontramos con que la herramienta que utilizamos no contiene todas las funciones o algoritmos que necesitamos o simplemente son complicadas de utilizar y requieren de un experto para poder utilizarlas, incluso en ciertos casos es necesario integrar múltiples herramientas para realizar las operaciones indispensables para el análisis.

El trabajo aquí propuesto desarrolla una herramienta de análisis de grafos como servicio, logrando esto a través de una infraestructura basada en la nube la cual permite trabajar con único sistema escalable y elástico al cual los usuarios podrán acceder a través de una API y así realizar diferentes operaciones de análisis a un grafo de una manera sencilla y sin necesidad de conocer particularidades de la operación, además este servicio puede ser consumido por cualquier aplicación que permita visualizar un grafo.

Para validar los resultados se ha utilizado un conjunto de datos de Twitter, pudiendo así realizar un análisis de redes sociales. Además, se ha creado una pequeña aplicación de Python y plotly, la cual invoca las funciones con el uso de la API y muestra los resultados obtenidos con ploty.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ACRÓNIMOS Y ABREVIATURAS

SNA		*Social Network Analisys*
CAVE		*Cave Automatic Virtual Environment*
LCD		*Liquid Cristal Display*
HMD		*Head Mounted Display*
UNHCR		*United Nations High Commissioner for Refugees*
TOG		Trabajo de Obtención de Grado
AWS		*Amazon Web Services*
VR		*Virtual Reality*
API		*Application programming Interface*
HITS		*Hypertext Induced Topic Selection*
BFS		*Bread First Search*
DFS		*Deep First Search*
INSNA		*International Network for Social Network Analysis*
ACID		*Atomicity, Consistency, Isolation and Durability*
HTTP		*Hyper Text Transfer Protocol*
EC2		*Elastic Cloud Computing*
CPU		*Central Processing Unit*
SQL		*Structured Query Languaje*
SDK		*Software development Kit*
CORS		*Cross Origin Resource Sharing*
WCC		*Weakly connected component*
SSH		*Secure Shell*
IAM		*Identity Acess Managment*
ITESO		Instituto Tecnológico y de Estudios Superiores de Occidente

1. INTRODUCCIÓN

En la actualidad para muchas instituciones tanto educativas como de ámbito privado es de gran interés el análisis de información masiva y sus relaciones, esta información puede ser representada a través de diferentes estructuras. Una de estas son los grafos, mismos que nos permiten modelar relaciones entre objectos de una manera práctica y visual, es por eso que actualmente en la academia se encuentran gran cantidad de investigación, herramientas y nuevas técnicas para la visualizan y análisis de los grafos.

En este trabajo se propone una API de servicios para el análisis y transformaciones de grafos, que pueda ser conectado a cualquier solución diseñada para la exploración y visualización de grafos. Desarrollando esto a través de una API en AWS con servicios, algoritmos y técnicas de análisis visual como centralidad, *PageRank*, detección de comunidades, sumarización entre otras. Esta API se encontrará conectada a una base de datos de grafos (neo4j) para el almacenamiento y aplicación de algunos algoritmos en los datos.

1.1. Antecedentes

Los grafos son unas de las estructuras más importantes de la computación moderna, ya que nos ayudan a modelar problemas de interrelaciones entre unidades que interactúan unas con otras, es por eso por lo que los podemos encontrar en diversas áreas como economía, aeronáutica, física, biología (análisis del ADN) matemáticas y otras áreas, por eso no es de extrañarse que la teoría de grafos se ha vuelto un tema tan importante, el cual en 2019 generó aproximadamente 264,000 resultados en Google Académico.

Una de estas áreas que ha tomado importancia últimamente es el análisis de redes sociales. Algunas como *Facebook Twitter* e *Instagram* pueden llegar a producir información de gran valor con un análisis correcto, es por eso por lo que hoy en día existen diferentes tipos de herramientas y técnicas que nos ayudan a analizar estos grafos y de esta manera conseguir información de valor. Desde el uso de nuevas técnicas de visualización con ayuda de dispositivos de realidad virtual, con el fin de representar el grafo de una manera más intuitiva para el usuario lo cual en estudios ha comprobado mejorar los tiempos en algunas tareas de análisis [1] [2] [3], así también el uso de herramientas como *gephi*, *pajek*, *IGraph*, *NetworkX*, que nos proporcionan un catálogo de funciones y algoritmos (centralidad, grado, *PageRank*, detección de comunidad, encontrar caminos, etc.) que podemos aplicar a nuestro conjunto de información para generar información de utilidad y transformaciones. Con base en lo anterior usar las herramientas y técnicas correctas es fundamental para el éxito de una investigación.

1.2. Justificación

Hoy en día es de gran interés tanto en la academia como en la industria generar valor de la información, ya que, gracias a *boom* de la tecnología, la cantidad de información que tenemos a la disposición es inmensa, es así como surge la necesidad de utilizar la teoría de grafos para solucionar problemas específicos, así como analizar esta información y generar valor.

Sin embargo, en ocasiones la información no puede ser interpretada de manera correcta debido a la sobrecarga de datos en la visualización 2D ya que algunas herramientas no proporcionan algoritmos de visualización 3D, o las herramientas para consultar y analizar esta información requieren de un experto que domine la herramienta utilizada.

De lo anterior surge la necesidad de proponer una nueva solución a las herramientas que existen, hoy en día existe una gran cantidad de trabajo respecto a este tema, pero la gran mayoría se enfoca en el uso de las herramientas actuales ya sea en base de datos de grafos, librerías o herramientas específicas, pero existe muy poco trabajo relacionado a servicios web o APIs con algoritmos de análisis que permitan aplicar algoritmos de análisis y visualización fáciles de integrar a nuestras herramientas de visualización sin necesidad de comprometernos con una herramienta en especifica o hacer múltiples interfaces de integración.

1.3. Problema

A pesar de que hoy en día existen diferentes herramientas para el análisis, visualización y exploración de grafos, no existe una solución completa o definitiva, ya sea por el hecho de que las soluciones actuales no proporcionen todos los algoritmos necesarios para la exploración y análisis o la herramienta no soporte una conexión a una base de datos de grafos, o no cuente con un método de visualización.

Es por lo que frecuentemente se necesita del uso de múltiples herramientas y/o expertos para el análisis de la información, lo cual dificulta y convierte el proceso de análisis en una tarea muy compleja de realizar, debido a que en ocasiones se requiere crear múltiples interfaces de integración entre las herramientas para realizar una transformación en el grafo.

Además de esto, ninguna de las herramientas cuenta con todas las funciones necesarias para trabajar en paradigmas tanto locales como globales, por ejemplo, ninguna de las herramientas antes mencionadas proporciona algoritmos o funciones para la sumarización de grafos basados en tipos de nodos.

1.4. Objetivos

1.4.1. Objetivo General:

Desarrollar una solución agnóstica para el análisis de datos basados en grafos, enfocándonos para esta prueba de concepto en el análisis de redes sociales en específico *Twitter*, pero diseñando la herramienta de manera que esta pueda a ser utilizada en cualquier tipo de grafo. La solución debe proveer de una web API para que pueda ser integrada fácilmente a cualquier plataforma o herramienta de visualización que utilizaremos para visualizar y manipular el grafo.

1.4.2. Objetivos Específicos:

- Utilizar un dispositivo o herramienta para consumir la información, generar subgrafos y aplicar filtros.
- Generar material de valor que sirva de referencia para futuros proyectos o trabajos de obtención de grado.
- Desarrollar una web API que pueda ser reutilizada en futuros trabajos y sea utilizada en cualquier tipo de grafos siempre y cuando pueda ser importada a una base de datos de grafos como neo4j.
- Incrementar mi conocimiento en la teoría de análisis de grafos, así como en el uso de la nube de AWS.
- Utilizar el conocimiento adquirido para generar valor en mi empresa con la creación de herramientas relacionadas con estructuras de grafos.

1.5. Novedad científica, tecnológica o aportación

Actualmente los servicios que se ofrecen en la nube son de gran utilidad para los desarrolladores pues les permiten crear herramientas y aplicaciones de una manera más rápida sin necesidad de ser expertos en cierto tema un ejemplo de esto son los servicios de reconocimiento de imágenes que ofrece Amazon, nuestra solución propone un prototipo de un sistema similar, pero para el análisis de grafo ya que actualmente no existe una solución de esta índole.

Este trabajo incluirá tecnologías de cómputo en la nube como plataforma e infraestructura de desarrollo combinada con técnicas y algoritmos de análisis y visualización de grafos, finalmente proporcionará todas estas funciones y algoritmos a través de una API la cual puede ser consumida a través de peticiones HTTP.

2. ESTADO DEL ARTE O DE LA TÉCNICA

En esta sección presentaremos diferentes publicaciones y estudios con los que se puede comprender las diferentes metodologías que existen para el análisis y visualización de grafos, así como las herramientas y técnicas utilizadas para ello.

2.1. Herramientas para el análisis de redes sociales

Las redes sociales representan una gran área de oportunidad de estudio es por eso por lo que cada vez crece el interés de estudiarlas. El análisis de redes sociales (SNA) es una estrategia de investigación de estructuras sociales usando redes y teoría de grafos [4], con el cual podemos descubrir estructuras dentro la red, encontrar comunidades, encontrar nodo influyentes, identificar tendencias, ranqueo de nodos, etc.

Cuando hablamos de herramientas de análisis de redes sociales nos referimos a herramientas para manipulación y visualización de grafos en su mayoría, herramientas como *gephi*, *pajek*, *Igraph*, *NetworkX*. De las cuales trabajos anteriores se han dedicado a comparar para determinar cuáles pueden tienen áreas de oportunidad.

Un primer trabajo de esta comparación fue realizado en 2010 por David Combe [5] el cual en años posteriores fue retomado y expandido por Nadeem Akhtar [6], en este trabajo se realiza un comparativa de las soluciones existentes bajo los siguientes criterios:

- (P) Comparación basada en plataforma
- (T) Comparación basada en tipo de red que soporta
- (V) Comparación basada en tipos de visualización o representación del grafo
- (F) Comparación basada complejidad temporal de los algoritmos, tipos de archivo que puede recibir como entrada, y funciones

Del trabajo anterior podemos resumir los puntos más relevantes en la tabla 2.3.

Tabla 2-1 Comparativa entre herramientas de SNA más populares [6]

	NetworkX	*IGraph*	*Pajek*	*Gephi*
Descripción	Es un paquete de Python para la creación, manipulación y estudio de redes complejas [7].	Colección de herramientas para el análisis de redes, disponible en Python, R and C/C++ [8].	Software principalmente para visualizar grafos, con capacidades de análisis [9].	Herramienta de visualización y exploración para grafos y redes [10].
(P) Tipo	Librería	Librería	Herramienta	Herramienta
(P)Número de nodos	1 millón	1 millón	0.15 millones	1 millón
(V) Aleatorio	Si	Si	Si	No
(V) *Graphviz*	Si	No	No	No
(V) *Kamanda kawai*	No	Si	Si	No
(V) *Fruchterman Reingold*	No	Si	Si	No
(V) *Forced Atlas*	No	No	Si	No
(F) Centralidad	Si (58.57s)	Si (6.19s)	Si (2s)	Si (4s)
(F) Modularidad	Si (81.4s)	Si (9s)	Si (6s)	Si (30s)
(F) Comunidades	Si	Si	No	Si
(F) *PageRank*	Si (120.78s)	Si (9.81s)	Si (2s)	Si (4s)
(F) *Partition*	No	No	Si	No
(F) *HITS*	Si (57.23s)	Si (15.43)	No	Si (8s)
(F) *BFS*	Si	Si	Si	No
(F) *DFS*	Si	Si	Si	No
(F) Sumarización	No	No	No	No

De la siguiente información podemos concluir que el uso de herramientas independientes como *gephi* o *pajek* no son lo más conveniente para nuestra solución ya que no serían fácil de integrar a pesar de ser muy cómodas para visualizar información, en cambio módulos como *NetworkX* o *Igraph* nos podrían ayudar a realizar operaciones, en especial *Igraph* ya que ofrece un mayor catálogo de funciones y es mucho más rápida. A pesar de que no posee la capacidad de hacer la sumarización podemos usar uso de otras de sus capacidades de ser necesario, ya sea para realizar operaciones en los datos de nuestra base de datos de grafo y encapsular todo a través de una API de servicios web sencilla de utilizar.

Otro trabajo propuesto por Marcin Mincer [11] se enfoca en herramientas de SNA para la investigación de relaciones sociales entre personas, para determinar conexión interpersonal en una red social, determinar la importancia de los nodos en la red y detectar comunidades de nodos. Los datos en esta investigación fueron conseguidos de Facebook y Twitter se usaron técnicas de minería de datos con el propósito principal de presentar métodos para conseguir información valiosa de plataformas de redes sociales comunes.

Un último trabajo que analizaremos sobre este tema es el propuesto por Navpreet Kaur [12] en el cual se desarrolla un *framework* para el análisis de datos de *Twitter*, usando la herramienta *NetworkX* y *Twitter*

API, lo novedoso de este estudio es que usa la API de *Twitter* para consumir los datos en tiempo real, después estos datos son procesados para convertir esta información en forma de lista a una estructura de grafo para después ser analizados usando algoritmos como centralidad, grado, centralidad de intermediación. Con esta información Navpreet Kaur es capaz de identificar, quien habla de quien, perfiles de usuario falsos y personalidades famosas entre los *retweets*.

2.2. Exploración, visualización y creación de sentido en grafos

En este trabajo se exploran diferentes técnicas para darle significado al grafo o comprender el grafo, es decir técnicas que nos ayudan a entender la información de mejor manera [13], donde de acuerdo con el autor todas la técnicas y herramientas se definen en dos categorías: operaciones locales y operaciones globales (ver tabla 2.4).

Tabla 2-2 Jerarquías de creación de sentido de grafos sdeacuerod a Robert Pienta [13]

Creación de sentido en grafos	Vista Globales		Filtrado
			Muestreo
			Partición
			Agrupamiento
	Vistas Locales	Libre Exploración	Exploración
			Redes Motifis
		Exploración con Objetivo	Coincidencia de Patrones
			Navegación

- Las metodologías globales las define como las operaciones donde se empieza con una vista general, después se enfoca en un punto de interés y se filtra, y finalmente se observan los detalles que se requieran. Una de las desventajas de este tipo de enfoque es que no es muy útil cuando se trabaja con grafos que contienen millones de nodos.
- Las metodologías locales se enfocan más en desplegar subgrafos o subconjuntos de información, lo cual llega a comprometer la visualización general del grafo, pero a cambio se requiere menos computación y se llega a tener una mejor compresión visual de la información.

De este trabajo podemos analizar y proponer nuestra metodología, un hibrido de estas técnicas donde se pueda visualizar el grafo en su totalidad y tener diferentes subgrafos a partir del agrupamiento de nodos donde después podamos utilizar técnicas de exploración visual local para identificar nodos de interés.

3. MARCO TEÓRICO/CONCEPTUAL

En este capítulo se presentan las bases teóricas y conceptuales sobre el análisis y visualización de grafos, sus aplicaciones y algoritmos necesarios para esto además herramientas necesarias para el desarrollo de nuestra propuesta.

3.1. Teoría de Grafos

La teoría de grafos se refiere a estudio de los grafos, los cuales son estructuras matemáticas usadas para representar objetos interconectados es decir su relación entre ellos [14]. Los grafos en general son unas de las estructuras más importantes en las ciencias computacionales debido a que muchos problemas de diferentes disciplinas se pueden modelar usando estas estructuras [15].

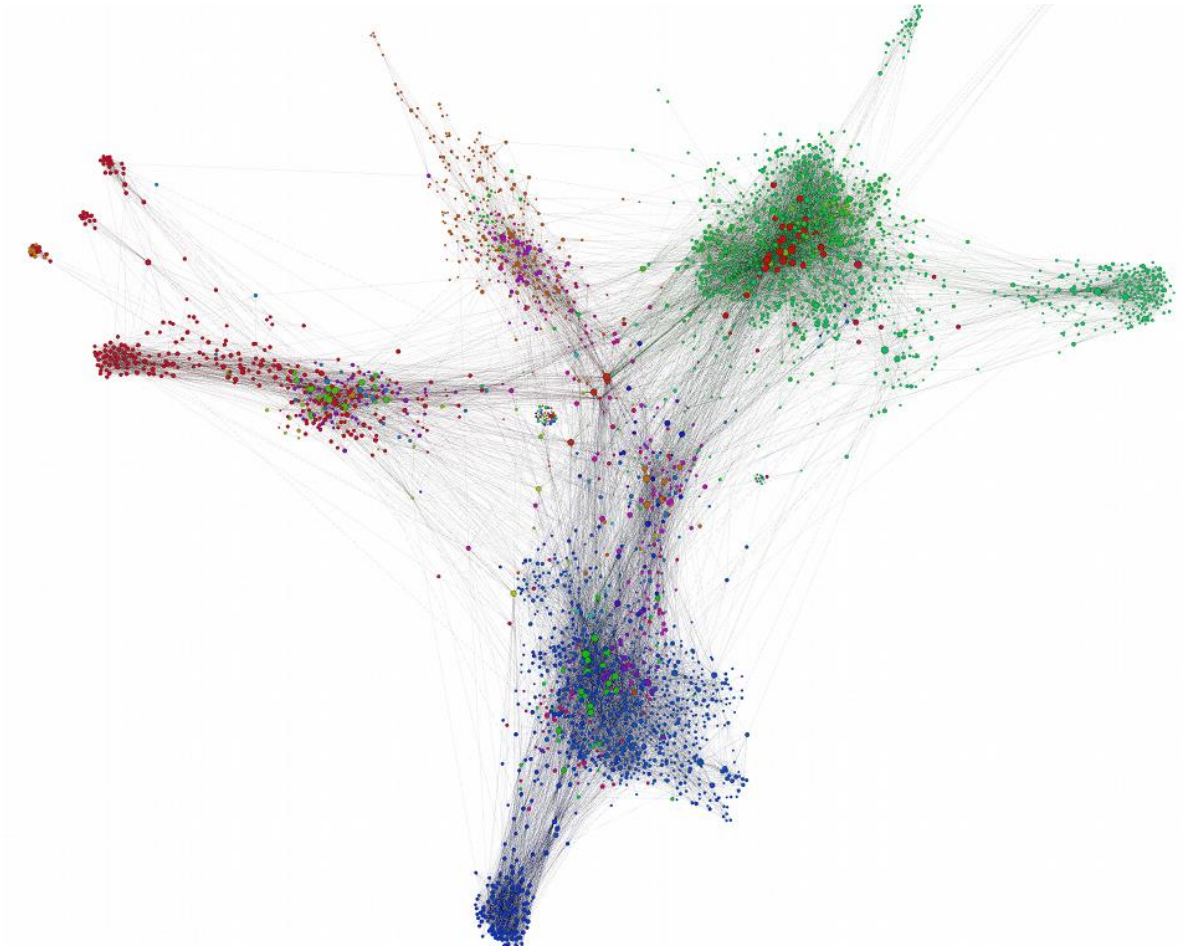

Figura 3-1 Ejemplo de un Grafo (visualización de un grafo usando gephi) [16]

Un grafo G = (V, E) está conformado de dos conjuntos V y E [17].

- Los elementos de V llamados vértices o nodos.
- Los elementos de E llamados aristas.

Nota: Las aristas son conectores entre vértices, es decir cada arista tiene un conjunto de uno o dos vértices asociados a ella.

- Un vértice u es adyacente a v si están conectados por una arista.
- Dos vértices adyacentes son llamados vecinos.
- Las aristas adyacentes son aquellas que están conectadas a un nodo en común.

Las aristas que sirven como conectores entre vértices pueden tener o no dirección, Cuando las aristas tienen dirección se dice que es un grafo dirigido en este caso el flujo es indicado con una flecha en caso de una representación de nodos y links, hay que tomar en cuenta que una misma arista puede tener flujo en ambos sentidos. Cuando no se tiene dirección se dice que es un grafo no dirigido (Figura 3.2).

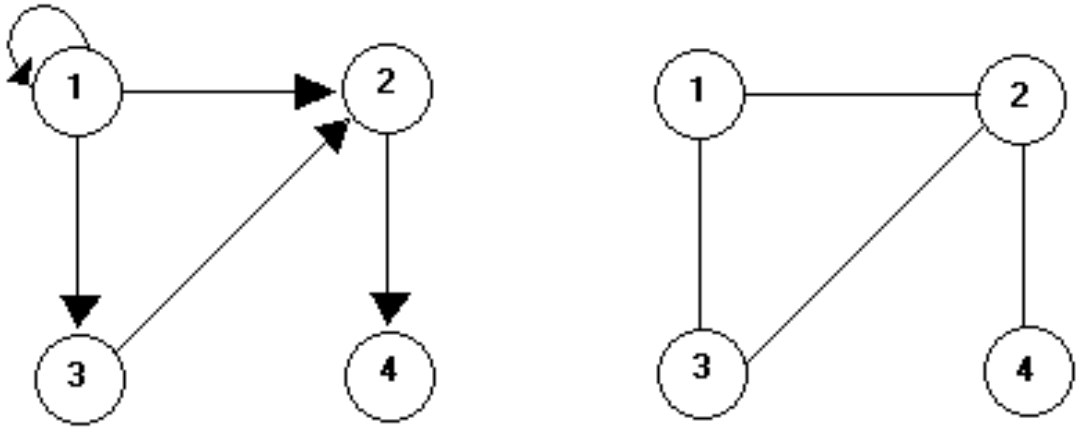

Figura 3-2 Ejemplo de grafo dirigido y uno no dirigido [18]

3.1.1. Aplicaciones de la teoría de Grafos

Los grafos los podemos encontrar en todos lados dentro de nuestra vida, simplemente no siempre podemos obsérvalos de manera directa, pero son una de las estructuras más dinámicas que hay, a continuación, se muestra ejemplos de donde se utilizan los grafos:

- Cuando navegamos dentro un buscador de una página de internet, en realidad estamos navegando un grafo, incluso el orden en que se nos muestran los enlaces tiene que ver con operaciones en el grafo que nos muestran el nodo más influyente de la red.
- En los sitios web de comercio electrónico como los son (*Ebay*, Mercado Libre, Amazon) las recomendaciones de artículos son resultado de operaciones en un grafo de artículos usando información de usuarios con gustos similares.
- En redes sociales como *Facebook*, *Twitter* o *Instagram* tenemos una red enorme donde cada usuario puede ser presentado con un nodo y cada arista puede ser representar las relaciones entre ellos, o en *Twitter* podemos usar un grafo dirigido para saber quién sigue a quien, y así obtener a los nodos más influyentes y mucho más, esto da a pie mucha investigación.

3.1.1.1. Análisis de redes sociales

El análisis de redes sociales se refiere a la técnica de investigación que se enfoca en identificar y analizar las relaciones sociales ente nodos de un grafo [6]. El análisis de redes sociales es un extracto de la teoría de grafos así que básicamente son los mismo solo que enfocado en redes sociales, básicamente las operaciones principales para lo que se utiliza son las siguientes:

- Descubrir la estructura social
- Encontrar valores y atributos de un nodo (centralidad, centralidad de intermediación, camino más corto, densidad, etc.)
- Encontrar comunidades
- Encontrar nodos influyentes

3.1.2. Algoritmos de Análisis, visualización y métricas asociadas

Al momento de analizar un grafo contamos con diferentes técnicas, métricas y algoritmos que nos facilitan la interacción con este en este capítulo se pretende revisar las más comunes al momento de hacer un análisis de grafos.

3.1.2.1. Centralidad

En la teoría de grafos los algoritmos de centralidad nos ayudan a determinar la importancia de distintos nodos dentro de la red. La centralidad de un nodo nos puede ayudar a determinar el impacto o influencia de una persona en una red social, la relevancia de un artículo en una red de información [19].

Debido a que la centralidad no es atributo intrínseco de los vértices, sino un atributo estructural esta siempre se mide bajo cierto criterio, por lo que tenemos diferentes tipos de centralidad, entre los más comunes para el análisis de redes sociales y nuestro proyecto se encuentran la centralidad de intermediación, de grado, de cercanía, de vector propio y *PageRank*.

3.1.2.1.1. Centralidad de Intermediación

La centralidad de intermediación es una manera de detectar la cantidad de influencia que tiene un nodo sobre el flujo de información en el grafo, es usualmente usada para encontrar los nodos que sirven de puente entre subgrafos [19].

El algoritmo de centralidad de intermediación calcula el camino más corto entre cada par de nodos conectados en el grafo usando *BFS*, después cada nodo recibe un puntaje basado en el número de veces en que el camino más corto utilizo este nodo [19].

Tabla 3-1 Ejemplo de centralidad de intermediación basado en la figura 3.3

Nombre del Nodo	Centralidad de Intermediación
Alice	4
Charles	2
Bridget	0
Michael	0
Doug	0
Mark	0

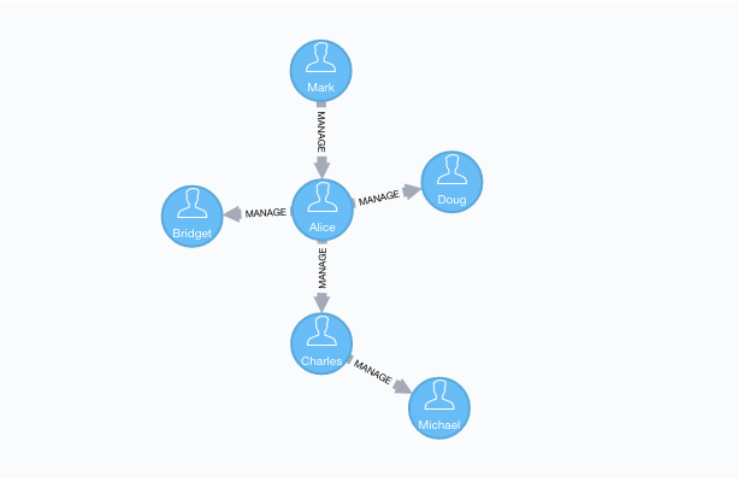

Figura 3-3 Ejemplo de Centralidad de intermediación en una red social [19]

3.1.2.1.2. Centralidad de Grado

La Centralidad de Grado o valencia de un vértice o nodo es el número de aristas incidentes en el vértice [30]; Y es usualmente usado para encontrar los nodos importantes o influyentes dentro una red social.

Cabe mencionar que cuando tenemos grafos dirigidos también podemos llegar a tener:

- *Indegree*: número de aristas con dirección hacia afuera de nodo.
- *Outdegree*: número de aristas con dirección hacia el de nodo.

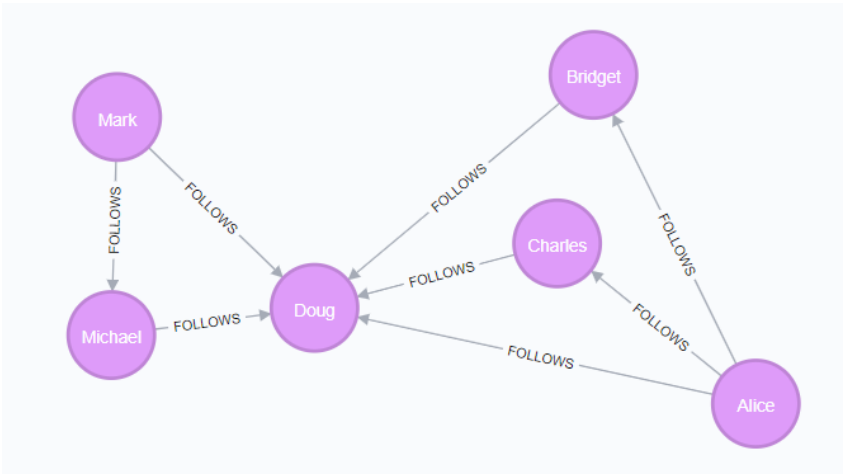

Figura 3-4 Ejemplo de red social para centralidad de grado usando neo4j

Tabla 3-2 Centralidad de grado de la figura 3-4

	Grado (*degree*)	*Indegree*	*Outdegree*
Doug	5	5	0
Alice	3	0	3
Bridget	2	1	1
Charles	2	1	1
Mark	2	0	2
Michael	2	1	1

3.1.2.1.3. Centralidad de Cercanía

La centralidad de cercanía nos ayuda a detectar nodos que son capaces de esparcir información a través de grafo, la cercanía es la lejanía promedio a todos los demás nodos, los nodos con alta cercanía tienen las distancias más cortas a todos los demás nodos [19].

Unos ejemplos del uso de la centralidad de cercanía son, estimar el tiempo que tomara a cierto mensaje para ser distribuido a través del grafo, estimar la importancia de una palabra en un documento [19].

3.1.2.1.4. Centralidad de Vector propio (Eigenvector)

La centralidad de vector propio es un algoritmo que mide la influencia transitiva o conectividad de los nodos, esta puede ser usadas en los mismos casos que cuando se requiere *PageRank*.

Como observamos en la Figura 3.5 el nodo *Home* cuenta con la mayor centralidad de vector propio porque tiene aristas entrantes de todos los demás nodos de la red, y no solo eso es importante sino la importancia de los nodos detrás de esas aristas.

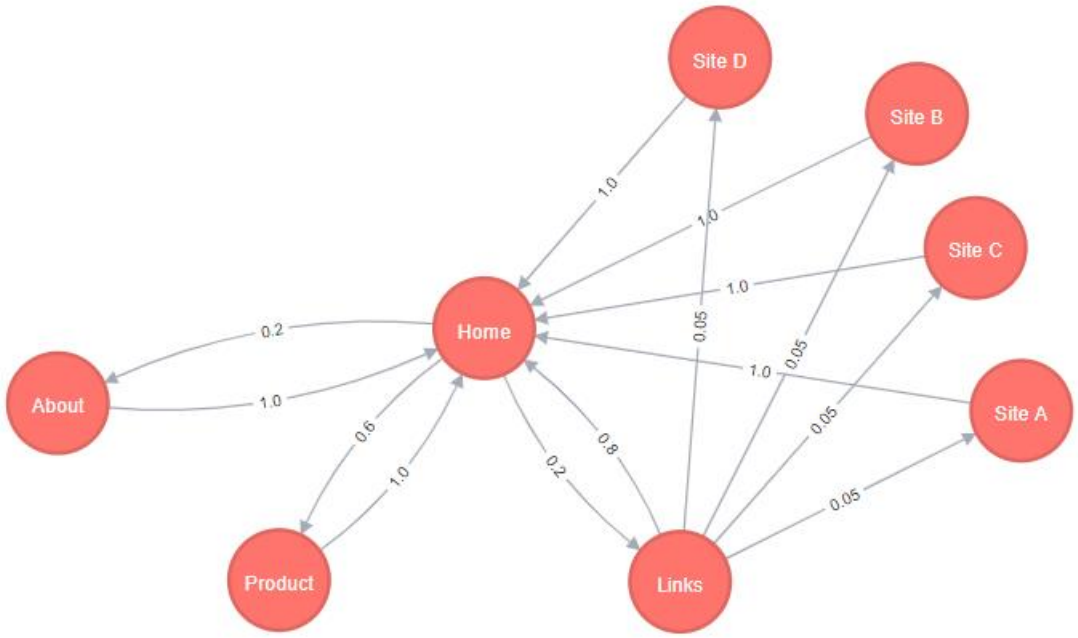

Figura 3-5 Grafo ejemplificando centralidad de vector propio [19].

3.1.2.1.5. Algoritmo *PageRank*

El algoritmo de PageRank al igual que la centralidad de vector propio es un algoritmo que mide la influencia transitiva de los nodos. *PageRank* fue nombrado por el cofundador de Google y fue originalmente usado para calificar las páginas web de Google.

3.1.2.2. Algoritmos de detección de comunidades

Las comunidades son grupos de vértices que probablemente comparten propiedades en común o juegan roles similares dentro de la red [20], la detección de comunidades es una tarea compleja identificar los nodos en común y sus bordes, es una tarea importante que en general nos ayuda a identificar jerarquías.

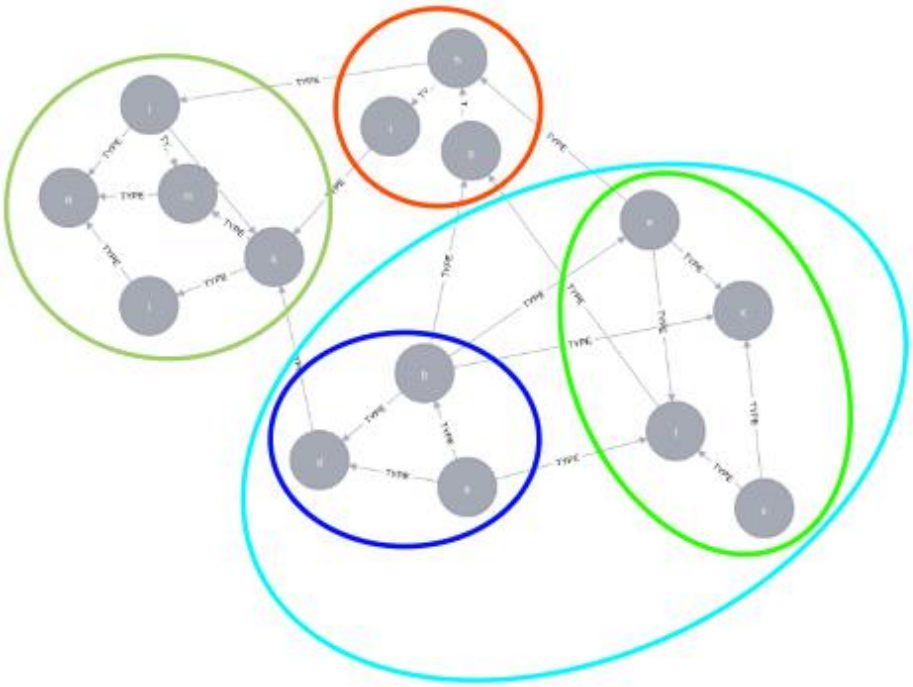

Figura 3-6 Grafo mostrando comunidades encontradas dentro del el mismo [19]

3.1.2.2.1. Louvain

El método Louvain para la detección de comunidades maximiza un puntaje de modularidad por cada comunidad, donde la modularidad cuantifica la calidad de asignación de nodos a una comunidad. Esto se logra evaluando que tan densamente están conectados los nodos de una comunidad, comparado con los que están fuera de la comunidad.

3.1.2.2.2. Componentes débilmente conectados (*WCC*)

WCC es un algoritmo que encuentra conjuntos de nodos conectados en un grafo no dirigido, donde todos los nodos de un mismo conjunto forman un componente conectado [19].

3.1.2.3. Algoritmos de Visualización

Los algoritmos de dibujado de grafos combinan métodos de la teoría de grafos y visualización de la información para poder construir representaciones de los datos en una manera que genere patrones visuales que el experto pueda observar y analizar, cabe mencionar que se pueden tener múltiples visualizaciones de un mismo grafo.

3.1.2.3.1. Force Directed

Los algoritmos *Force Directed* son de los algoritmos más flexibles para el dibujado de grafos, el propósito de es posicionar todos los nodos del grafo en un espacio de tres o dos dimensiones de manera que todas las aristas se encuentren lo más cercano a una misma distancia promedio y se crucen la menor cantidad de aristas posibles [21].

El algoritmo asigna fuerzas entre las aristas y vértices del grafo basado en sus posiciones relativas y después usa esas fuerzas para simular el movimiento de las aristas y nodos para minimizar su energía.

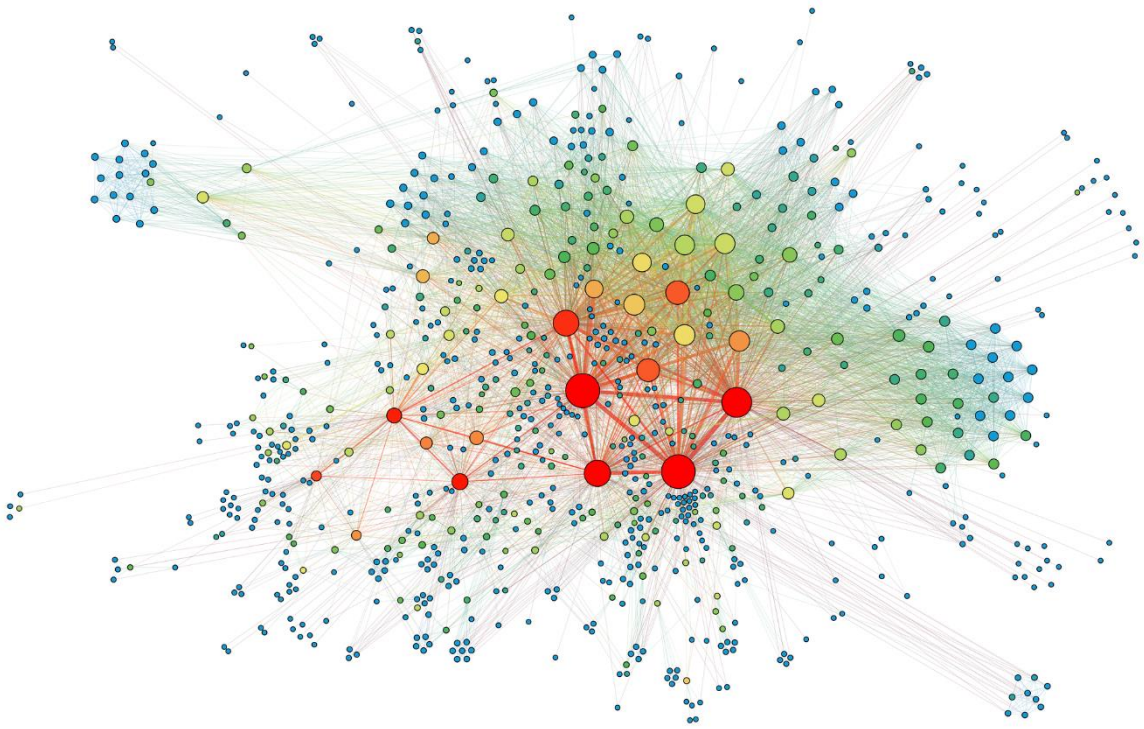

Figura 3-7 Red social visualizada usando un algoritmo *force Directed* [22]

3.1.2.3.1.1. Fruchterman Reingold

El algoritmo Fruchterman Reingold es un algoritmo del tipo *Force Directed*, la idea principal de este algoritmo es considerar los nodos como aros de acero y las aristas como muelles entre ellos, donde la fuerza de atracción es análoga a la fuerza del resorte y a la fuerza repulsiva es análoga a la fuerza eléctrica. La idea básica es minimizar la energía del sistema moviendo los nodos y cambiando las fuerzas entre ellos [23].

En este algoritmo la suma de fuerzas del vector determinara la dirección en la que se desplazara el nodo, la anchura del paso es constante, y esta determina que tanto se puede mover un nodo en una iteración, cuando la energía del sistema es minimizada los nodos dejaran de moverse y el sistema alcanzara un estado de equilibrio o simplemente se alcanzaron el número de iteraciones especificadas.

3.2. Herramientas para el análisis y visualización de grafo

Las herramientas de análisis de redes son usadas para identificar, analizar, visualizar o simular nodos y aristas de varios tipos [6]. Existen múltiples herramientas para el análisis de grafos una gran cantidad de ellas las podemos encontrar listadas en el INSNA, para nuestro caso solo nos enfocaremos en dos Igraph y neo4j, estas herramientas nos facilitaran la manipulación de los datos además de proveernos con algoritmos para facilitar las transformaciones den el grafo y así potenciar nuestra API.

3.2.1. *Igraph*

Igraph es una colección de paquetes de software creada para la manipulación de grafos y análisis de redes. Sus módulos principales están escritos en C/C++ y tiene interfaces en R y Python [8].

3.2.2. Neo4j

Neo4j es una base de datos para grafos, que pertenece a la familia de las bases NoSQL que provee de transacciones que cumplen con las propiedades ACID [24]. Neo4j es usualmente referida como una base de datos de grafos nativos esto debido que los datos son almacenados directamente. Y la base de datos usa punteros para navegar el grafo.

Unas de las funcionalidades por lo que Neo4j es una de las bases de datos favoritas son:

- *Cypher*: Es un lenguaje para consultas declarativo muy fácil de usar similar a SQL, pero optimizado para grafos.
- *Drivers*: Cuenta con interfaces para diferentes leguajes como Java, Javascript, .NET, Python y otros.
- Flexibilidad: Los esquemas del grafo se pueden adaptar en tiempo real, es decir se pueden crear nuevos tipos de nodos y relaciones en demanda.
- Algoritmos y funciones: Neo4j nos provee con funciones y algoritmos que podemos utilizar en nuestro grafo como *Louvain*, *PageRank*, centralidad, detección de comunidades y otros.
- Interface de visualización: Neo4j nos proporciona una interfaz de visualización en un ambiente en la cual nos permite visualizar y manipular la información de una manera agradable.

3.3. Twitter

Twitter es una red social red enfocada en propagar información y noticias donde las personas se comunican a través de mensajes cortos de 140 caracteres llamados tweets (estos mensajes mayormente están compuestos de texto plano) con la intención de que las personas que te siguen los lean.

Twitter usa dos métodos principales para distribuir sus mensajes:

- *Tweet*: Donde *tweeting* se refiriere al acto de publicar información del usuario origen hacia sus seguidores (Figura 3.2).
- *Retweet*: este método es usado para compartir información que fue publicada por otro usuario, con la opción de agregar comentarios (Figura 3.2).

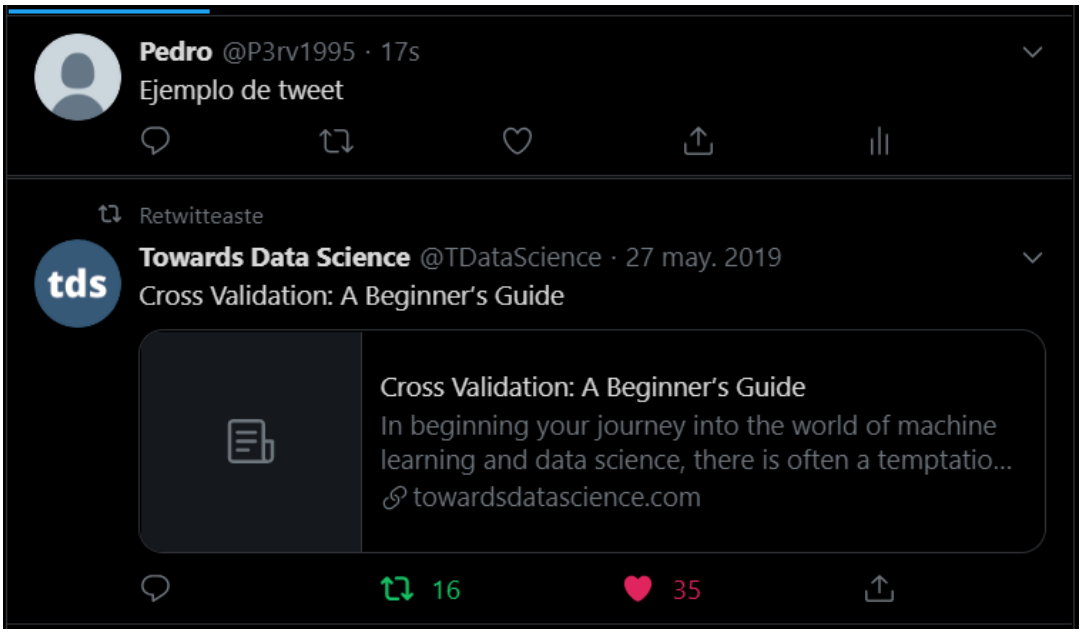

Figura 3-8 Ejemplo de tweet y Retweet

Anexado a esto Twitter incorpora las funcionalidades de #tag y @tag, donde #tag usa este método para generar tendencias y temas, por ejemplo, si un usuario quiere que su información se relacionada con cierto tema simplemente tendrá que agregar #tema, de esta manera si un usuario desea ver información relacionada sobre cierto tema solo seleccionar el #tag deseado, mientras @tag es una funcionalidad par etiquetar usuarios.

3.4. Computo en la nube

El Computo en la nube se refiere a el ofrecimiento de servicios de cómputo escalables como almacenamiento, base de datos, *software* de análisis y otros a través del internet [25].

Los principales beneficios del cómputo en la nube son:

- Costo: se elimina la necesidad de invertir en capital, compra de equipo e infraestructura.
- Velocidad: la mayoría de los servicios se ofrecen en demanda, en cuestión de un click puedes obtener los recursos necesarios
- Escalabilidad: habilidad de escalar elásticamente, esto quiere decir que se entregan la cantidad de recursos (almacenamiento, poder de procesamiento, bando de ancha) necesarios dependiendo de la necesidad en tiempo real.
- Desempeño: la mayoría de los proveedores de servicios en la nube cuentan con centros de datos en múltiples partes del mundo y con los equipos más actuales, lo cual reduce la latencia y asegura un desempeño óptimo.
- Seguridad: cuentan con políticas de seguridad para proteger tus datos y la información procesada o almacena en ellos.

3.4.1. Tipos de Servicios

Actualmente existen diferentes modelos de cómputo, como Software como servicios (SaaS), Plataforma como servicios (PaaS) e Infraestructura como servicio (IaaS), estos son comúnmente llamados como el *stack* del cómputo en la nube ya que permiten a los usuarios usar estos servicios para construir su solución.

3.4.1.1. Infraestructura como servicio

Ese es el tipo de servicio más básico que existe consiste en los bloques básicos de computación como espacio de almacenamiento, máquinas virtuales dedicadas, conexión de red y contexto de datos seguros [31].

3.4.1.2. Plataforma como servicio

Una plataforma como servicio nos ofrece la infraestructura informática necesaria a nivel de plataforma tales como servidores, sistemas de gestión de base de datos, kits de desarrollo de software (SDK), lo cual nos ayuda a enfocarnos solamente en el desarrollo y manejo de nuestras aplicaciones [31].

3.4.1.3. Software como servicio

El Software como servicio nos provee de un producto completo que es ejecutado y manejado por el proveedor del servicio, estos servicios pueden ser servicios de inteligencia artificial, de reconocimiento facial y otros.

3.4.2. Amazon Web Services (*AWS*)

AWS es uno de los proveedores de servicios en la nube más importantes del mundo, debido a que cuenta con el catálogo de servicios y funciones más grande que cualquier otro proveedor. AWS cuneta con servicios como computo, almacenamiento, Aprendizaje automático, inteligencia artificial e internet de las cosas [26]. Para nuestro proyecto decidimos usar AWS como plataforma para hospedar nuestra solución.

3.4.2.1. EC2

EC2 se refiere al servicio de cómputo en la nube ofrecido por AWS, el cual proporciona capacidad computación escalable en la nube de AWS y elimina la necesidad de invertir en hardware, lo que nos permite desarrollar e implementar aplicaciones en menor tiempo. EC2 nos permite escalar hacia arriba o hacia abajo para controlar cambios en los requisitos o picos de popularidad [27].

Características de EC2:

- Plantillas preconfiguradas.
- Configuraciones de CPU, memoria, almacenamiento y red.
- Volúmenes de almacenamiento para datos temporales que se eliminan cuando la instancia se detiene.
- Múltiples ubicaciones físicas para reducir la latencia.
- Direcciones IPv4 estáticas
- Redes virtuales para aislar y proteger instancias del resto de la nube

3.4.2.2. Lambda

AWS Lambda es un servicio sin servidor que ejecuta código en respuesta a eventos y administra automáticamente los recursos informáticos subyacentes. Este puede ser usado para crear servicios de *back-end* o para responder solicitudes HTTP a través de Amazon API Gateway. Además, puede ser conectado a otros servicios de la nube de Amazon como EC2 [28].

Cada función Lambda está compuesta de código, pruebas y cierta información de configuración como recursos que utiliza y eventos de activación, las lambdas pueden usar cualquier librería de terceros y son compatibles con múltiples lenguajes de programación como Java, Go, PowerShell, Node.js, C#, Python y Ruby [28].

La ventaja de usar lambdas es que no requerimos de un servidor, lo cual nos permite enfocarnos en el desarrollo del código, además que los recursos son escalados automáticamente de ser necesario dependiendo la cantidad de solicitudes y en cuestión de precio las lambdas solo se pagan por la duración de la ejecución en lugar de hacerlo por unidad de servidor que se utilizaría para hospedar nuestro código.

3.4.2.3. API Gateway

El servicio de API Gateway se encarga de la creación, manipulación, mantenimiento, monitoreo, y protección de la API, las API actúan como la entrada para que las aplicaciones puedan acceder a los datos o funciones, API Gateway gestiona todas las tareas implicadas con la aceptación y procesamiento de las llamadas a la API, como la administración de tráfico, compatibilidad con el CORS, control de acceso, administración de versiones [29].

4. ARQUITECTURA PROPUESTA

La solución consiste en una API de servicios web para manipular y operar grafos, la cual se comunica a través del protocolo HTTP y devuelve resultados en formato JSON, esta API Gateway de servicios está conectada a funciones lambdas (funciones sin servidor), estas funciones son las encargadas de ejecutar el código para realizar distintas operaciones con el grafo además de controlar y conectarse con el servidor que hospeda nuestro gestor de base de datos neo4j.

El servicio será consumido por una aplicación de Python capaz de visualizar y manipular la información, pero debido a que la solución está diseñada como servicio esta puede ser utilizada o integrada a distintas herramientas de visualización y manipulación de grafos que se puedan conectar a internet.

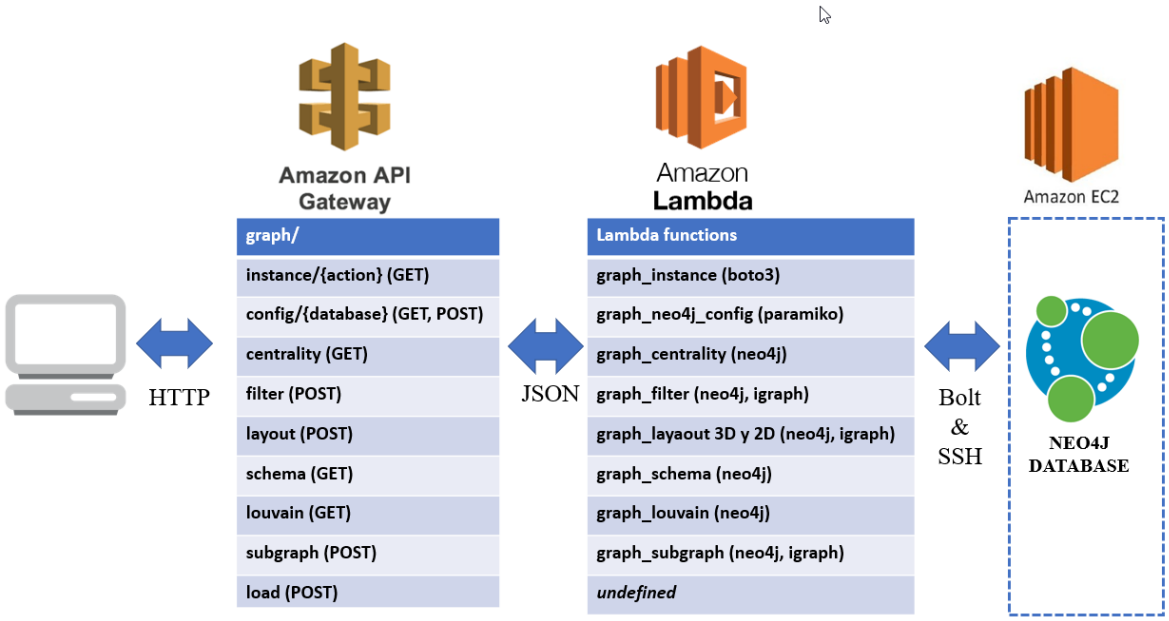

Figura 4-1 Diagrama General de la solución de análisis

4.1. Descripción de los datos

Para la prueba concepto de nuestra API usaremos los siguientes datos de Twitter; En nuestra base de datos contamos con 302,370 nodos y 781,394 aristas.

4.1.1. Esquema de datos, tipos y atributos

El esquema de nuestra base de datos esta descrito de manera sencilla en la siguiente imagen (Figura 4.1).

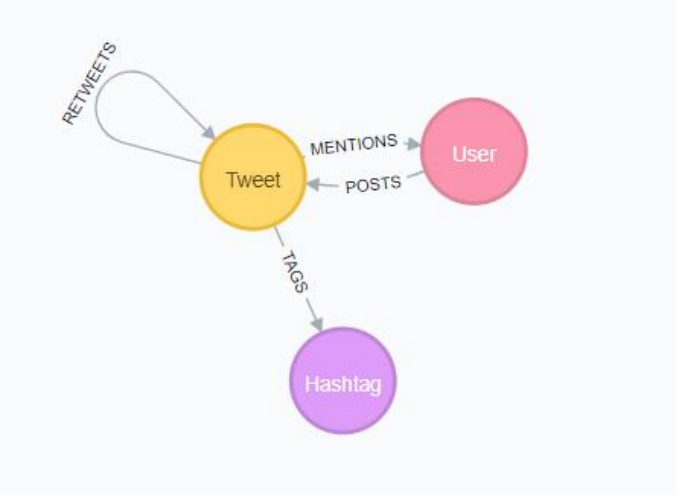

Figura 4-2 Esquema de la base de datos de Twitter para nuestra API

Tipos de datos:

- Vértices
  - *User*: usuario dentro de la red social Twitter.
  - *Tweet*: mensaje publicado por un usuario.
  - *Hashtag*: tema al que se puede asociar un mensaje.
- Aristas
  - *POSTS*: relaciona el usuario que origina un Tweet.
  - *MENTIONS*: relaciona si el Tweet hace mención o etiqueta a un usuario.
  - *RETWEETS*: indica que un Tweet hace mencione de otro Tweet.
  - *TAGS*: indica que en Tweet pertenece a cierto tema (Hashtag).

Propiedades:

Tabla 4-1 Propiedades de los nodos y aristas

Propiedad	Tipo de dato	*User*	*Tweet*	*Hashtag*
contributors	S		X	
contributors_enabled	B	X		
coordinates	S		X	
created_at	S	X	X	
dateFull	N	X	X	
dateShort	N	X	X	

dateTime	N	X	X	
day	N	X	X	
default_profile	B	X		
default_profile_image	B	X		
description	S	X		
display_text_range	S		X	
favorite_count	N	X	X	
favorited	B		X	
favourites_count	S	X	X	
follow_request_sent	S	X		
followers_count	N	X		
Following	S	X		
friends_count	N	X		
full_text	S		X	
geo	S		X	
geo_enabled	B	X		
has_extended_profile	B	X		
id_str	S	X	X	
in_reply_to_screen_name	S		X	
in_reply_to_status_id	N		X	
in_reply_to_status_id_str	S		X	
in_reply_to_user_id	N		X	
in_reply_to_user_id_str	S		X	
indices	S	X		
is_quote_status	B		X	
is_translation_enabled	B	X		
is_translator	B	X		
lang	S	X	X	
listed_count	N	X		
location	S	X		
message	S		X	
month	N	X	X	
name	S	X		
notifications	S	X		
place	S		X	
possibly_sensitive	B		X	
profile_background_color	S	X		
profile_background_image_url	S	X		
profile_background_image_url_https	S	X		
profile_background_tile	B	X		
profile_banner_url	S	X		
profile_image_url	S	X		
profile_image_url_https	S	X		

profile_link_color	S	X		
profile_sidebar_border_color	S	X		
profile_sidebar_fill_color	S	X		
profile_text_color	S	X		
profile_use_background_image	B	X		
protected	B	X		
quoted_status_id	N		X	
quoted_status_id_str	S		X	
retweet_count	N		X	
retweeted	S		X	
screen_name	S	X		
source	S		X	
statuses_count	N	X		
text	S		X	X
time_zone	S	X		
translator_type	S	X		
truncated	B		X	
uid	N	X	X	
url	S	X	X	
utc_offset	B	X		
verified	S	X		
year	N	X	X	

Nota: (S) String, (N) Number, (B) Boolean, X indica si el objeto cuenta con la propiedad

4.2. Almacenamiento de los datos

Para el almacenamiento de los datos usaremos una instancia de EC2 en conjunto con una base de datos neo4j, para ello usaremos la AMI de AWS Marketplace "Neo4j Graph Database - Community Edition".

El usar una instancia de EC2 nos proporciona una manera eficiente de almacenar la información ya que no necesitamos de hardware específico para una tarea sencilla como solo almacenar datos y tener dos puertos abiertos que necesitara neo4j, además las instancia permanecerá apagada la mayoría del tiempo lo cual genera un costo mínimo de $0.10 dólares Gigabyte por mes [30]por el Volumen de *SSD* y cuando esta instancia esta encendida el costo es de $0.10 dólares por hora [31].

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
m4.large	6.5	2	8	EBS only	Yes	Moderate

Figura 4-3 Descripción de la EC2 usada

Para gestionar los datos usamos neo4j, una poderosa base de datos de grafos la cual cuenta con un lenguaje (*Cypher*) sencillo y muy poderoso para manipular la información, además neo4j cuenta con algoritmos y funciones que podemos disponer y aplicar a nuestro grafo, lo cual nos ahorra tiempo en la implementación de algoritmos conocidos, por último, esta nos ofrece un módulo de Python para poder conectarnos y trabajar con ella fácilmente.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)
Root	/dev/sda1	snap-04eadbd1e777d0550	200	gp2	600 / 3000	N/A

Figura 4-4 Descripción del tipo de memoria usada por la EC2

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	0.0.0.0/0	
Custom TCP Rule	TCP	7473	0.0.0.0/0	
Custom TCP Rule	TCP	7687	0.0.0.0/0	
Custom TCP Rule	TCP	7474	0.0.0.0/0	

Figura 4-5 Puertos necesarios para neo4j (7474, 7473, 7687) y SSH (22)

En las Figuras 4.3, 4.4 y 4.5 se describe la configuración utilizada para la creación de la instancia de Amazon.

4.3. Desarrollo de la API

En esta sección describiremos la construcción de la API y como esta está conectada a las lambdas, la API está construida de 2 elementos principales: las funciones lambdas, las cuales simplemente están encargadas de ejecutar nuestro código, sin tener que preocuparnos por los recursos informáticos subyacentes y la API Gateway que se encarga de administrar, proteger, publicar y mapear las entradas de variables hacia nuestras lambdas.

4.3.1. Funciones Lambda

De igual manera se decidió usar funciones lambas para ejecutar el código ya que de esta manera no necesitamos de un hardware específico para nuestro código, también nos evita la configuración para nuestro lenguaje de programación. De igual manera el costo por usar estas lambdas es muy accesible, en promedio $0.20 dólares por 1 millón de ejecuciones, $0.000,016,667 dólares por Gigabyte segundo [32].

Para desarrollar decidimos usar Python 3.6, esto por las ventajas y facilidad en el desarrollo que nos permite el lenguaje, además que existen múltiples módulos o drivers que facilitan la manipulación de elementos de AWS, así como un módulo para trabajar con neo4j.

4.3.1.1. Módulos de Python principales

- Boto3: es el SDK de AWS para Python. Nos permite crear, configurar y administrar instancias de EC2 y S3, este módulo nos permitirá inicia, apagar y reiniciar la instancia donde se encuentra nuestra base de datos.

- Paramiko: nos permite establecer una comunicación con protocolo SSH, dándonos tanta funcionalidad de cliente y servidor [33], esto nos permite conectarnos a la instancia EC2 y editar el archivo de configuración de neo4j para cambiar de base de datos, además de que con el podemos listar las bases de datos disponibles en la instancia.
- Neo4j: este módulo es una interfaz que nos permite conectarnos a una base de datos neo4j a través del protocolo BOLT y además nos permite interactuar con ella usando lenguaje Cypher, para crear consultas modificar valores y leer resultados.
- Igraph: es una colección de herramientas para el análisis de redes sociales y grafos, este módulo será complementario a la funcionalidad y algoritmos integrados de neo4j, lo cual nos ahorrará tiempo de desarrollo para generar algoritmos de visualización.
- Json: este módulo nos permite manipular objetos JSON de una manera sencilla, su principal función dentro de nuestro código es empaquetar las respuestas cuando nuestro servicio sea invocado.

4.3.1.2. Funciones Lambda

- **graph_instance**

Tabla 4-2 Definición de la función graph_instance

Descripción	Esta función se encarga de prender, apagar, reiniciar, y entregarnos el estado de la instancia que hospeda nuestra base de datos.
Parámetros	*action* (*string*): palabra que define la acción a ejecutar (*start, stop, reset, status*).

Respuesta (JSON)	{
	"*response*": (*string*) descripción de la acción ejecutada.
	"*status*": (*number*) 0 si la ejecución fue un éxito 1 si hubo un error.
	}

- **graph_neo4j_config**

Tabla 4-3 Definición de la función graph_neo4j_config

Descripción	Esta función tiene dos modos de ejecución POST y GET, si se utiliza POST se necesita enviar el nombre de la base de datos a utilizar en una variable POST y la función se encargará de modificar el archivo de configuración de neo4j y seleccionar la base de datos especificada, si se utiliza GET se devolverá una lista con las bases de datos que tenemos disponibles.
Parámetros	*database* (*string*): nombre de la base de datos a utilizar.

Respuesta (JSON)	{
	"*db*": (*string*) base de datos seleccionada.
	"*db_list*": (*array<string>*) lista con las bases de datos disponibles en el sistema.
	"*status*": (*number*) 0 si la ejecución fue un éxito 1 si hubo un error.
	}

- **graph_schema**

Tabla 4-4 Definición de la función graph_neo4j_schema

Descripción	Regresa el esquema de la base de datos y sus propiedades.
Parámetros	No
Respuesta (JSON)	{ "*nodes*": (*array<string>*) lista con los tipos de nodos.

	"*edges*": (*array<string>*) lista con los tipos de aristas.
	"*properties*": {
	"*property_key*": "*TYPE*", (ejemplos a continuación)
	"*uid*": "*INTEGER*",
	"*in_reply_to_status_id_str*": "*STRING*", ……}
	}

- **graph_layout3D y graph_layout2D**

Tabla 4-5 Definición de la función graph_layout3D y graph_layout2D

Descripción	Aplica el algoritmo de visualización Fruchterman Reingold (una variación de los algoritmos tipos force Directed) al grafo, al aplicar esta función se agregarán 3 atributos más (x, y, z) a todos los nodos del grafo.

Parámetros (JSON)	{
	"*attraction_multiplier*": (*number*)
	"*repulsion_multiplier*": (*number*)
	"*max_iterations*": (*number*)
	"*width*": (*number*)
	"*height*": (*number*)
	"*epsilon*": (*number*)
	}

Respuesta (JSON)	{
	"*status*": (*number*) 0 si la ejecución fue un éxito 1 si hubo un error.
	"*desc*": (*string*) descripción del error en caso de existirlo.
	}

- **graph_centrality**

Tabla 4-6 Definición de la función graph_centrality

Descripción	Aplica la centralidad del grado del grafo, añadiendo 3 atributos nuevos (*indegree*, *outdegree*, *degree*) a cada nodo del grafo, estos atributos contienen los grados de los nodos.
Parámetros	NO

Respuesta (JSON)	{
	"*status*": (*number*) 0 si la ejecución fue un éxito 1 si hubo un error.
	"*desc*": (*string*) descripción del error en caso de existirlo.
	}

- **graph_louvain**

Tabla 4-7 Definición de la función graph_louvain

Descripción	Aplica el algoritmo de *louvain* al grafo, el cual agrega 2 atributos nuevos al grafo (*community* y *communities*), comunidad, el cual es un número que representa la comunidad principal del grafo y comunidades el cual es una lista con todas las comunidades al que pertenece el nodo.
Parámetros	NO

Respuesta (JSON)	{
	"*status*": (*number*) 0 si la ejecución fue un éxito 1 si hubo un error.
	"*desc*": (*string*) descripción del error en caso de existirlo.
	}

- **graph_filter**

Tabla 4-8 Definición de la función graph_filter

Descripción	Esta función nos permite aplicar diferentes filtros al grafo de una manera sencilla sin necesidad de conocer algún lenguaje de consultas de grafo, la información la podemos filtrar ya sea por sus atributos o tipos.

Parámetros (JSON)	{
	"*limit*": (*number*) límite de nodos a devolver por la función.
	"*graph*": (*Graph*) este es un parámetro opcional en caso de que se quiera trabajar con un grafo diferente al almacenado en la base de datos la única condición es que debe conocer el esquema del grafo con el que se desea trabajar ya que si algún tipo o atributo es incorrecto ocurrirá un error.
	"*filters*": (*array<Filters>*) lista de filtros a aplicar al grafo.
	"*unions*": (*array<string>*) los únicos *strings* validos son "*and*" y "*or*" que representan la intersección y unión de filtros respectivamente. Es importante notar que este parámetro unirá los filtros que tenemos por lo que siempre debemos enviar n-1 uniones por cada n filtros, en caso de enviar una cantidad incorrecta producirá un error.
	}

Respuesta (JSON)	{
	"*status*": (*number*) 0 si la ejecución fue un éxito 1 si hubo un error.
	"*graph*": (*Graph*) objeto grafo.
	}

La siguiente tabla contiene la descripción de los objeto**s** utilizados por la función graph_filter

Tabla 4-9 Descripcion de objetos utilizados por graph_filter

Objetos	**Descripción de la estructura**

Graph	{
	"*nodes*": (*array<object>*) lista de objetos donde cada objeto representa un vértice y cada objeto está compuesto de los atributos del nodo.
	"*edges*": (*array<object>*) lista de objetos donde cada objeto representa una arista y cada objeto está compuesto de los atributos de la arista.
	}

Filter	{
	"*element*": (*string*) elemento al que se le aplicara el filtro "v" para vértice "e" para arista, este atributo es obligatorio.
	"*element_type*": (*string*) tipo o clase al que se le aplicara el filtro, en nuestro caso tenemos *Twitter*, *Hashtag* y *User* para vértices y *post*, *mention*, *tag* para aristas. Este atributo es obligatorio.
	"*attr*": (*string*) atributo al que se le aplicara el filtro, este parámetro es opcional
	"*operator*": (*string*) operador a utilizar; Este atributo es obligatorio si se usa *attr*
	"*data*": (*string*, *number*, *array<string*, *number>*) datos a usar con el operador. Este atributo es obligatorio si se usa *attr*.
	}

Los operadores varían dependiendo el tipo de dato a utilizar (Tabla 4.10).

Tabla 4-10 Operadores válidos para el objeto *Filter*

String	**Stars_with**: la cadena debe comenzar con los caracteres indicados. **Ends_with**: la cadena debe terminar con los caracteres indicados.

	=: la cadena debe tener los caracteres indicados ¡=: la cadena debe ser distinta a los caracteres indicados. **Contains**: la cadena contiene los caracteres indicados.
Number	¡=: número distinto. =: número idéntico. <: número menor que. <=: número menor o igual que. <: número mayor que. <=: número mayor o igual que. **Betwen**: número entre los parámetros indicados. **In**: número se encuentra en la lista.

- **graph_subgraph**

Tabla 4-11 Definición de la función graph_subgraph

Descripción	Esta función nos entrega un subgrafo basado en la sumarización de nodos del grafo de la base de datos. La sumarización es basada en tipos de nodo. Todos los atributos entre el nodo origen y nodo destino serán almacenados en la nueva relación que une los nodos, además esta relación será asociada a un tipo nuevo el cual su nombre será determinado por todos los vértices y aristas que estén en el camino del nodo inicial y el nodo destino. (ver la figura 4.6)
Parámetros	*Node_type* (*string*): tipo de nodo a sumarizar.

Respuesta (JSON)	{
	"*status*": (*number*) 0 si la ejecución fue un éxito 1 si hubo un error.
	"*graph*": (*Graph*) objeto grafo.
	}

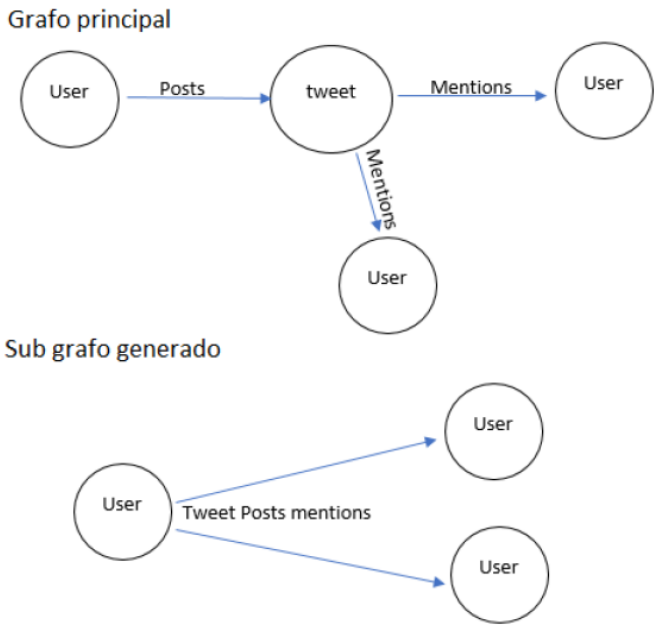

Figura 4-6 Demostración de sumarización de un grafo por tipo *User*

4.3.2. Diseño del la API Gateway

Para hacer uso de nuestras funciones ahora necesitamos una manera de hacerlas públicas, es aquí donde entran las API Gateway ya que nos permiten mapear nuestro código a un punto de entrada y salida de datos en la red. Además, Amazon nos proporciona una interfaz fácil de utilizar para mapear tanto nuestras lambdas como los parámetros requeridos por nuestras funciones. Además, el costo es muy accesible pues la llamada a la API es de $1.00 dólar por millón de llamadas [34].

La estructura de nuestra API se encuentra descrito en la parte izquierda de la imagen 4.7. Como podemos observar, se encuentran las rutas de la API y métodos de acceso, además, cada método está conectado a una lambda la cual tiene el mismo nombre de la API, como observamos en la parte derecha de esta misma figura.

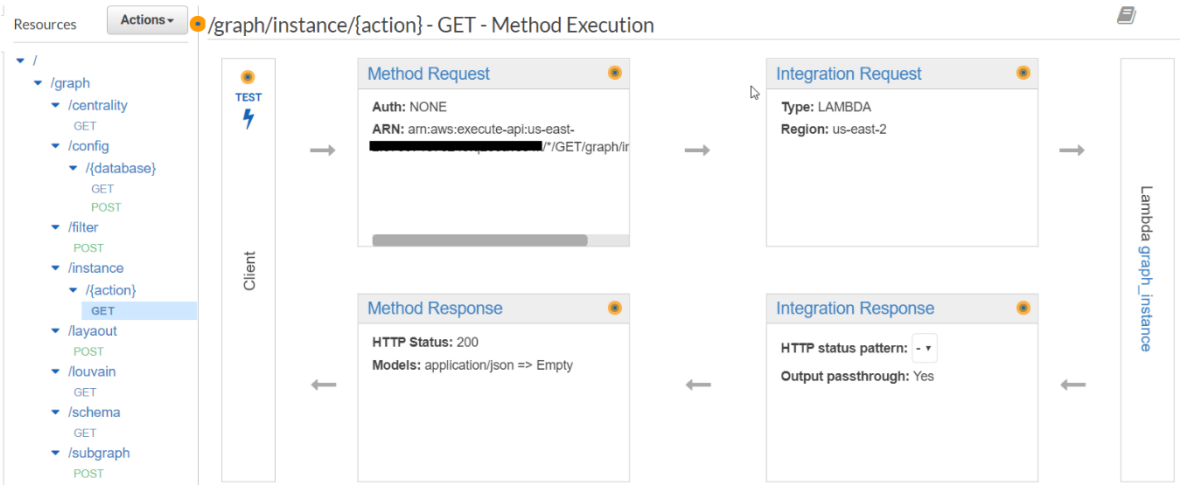

Figura 4-7 Estructura y mapeo de la API, y ejemplo de configuración para graph_instance lambda

Además de mapear nuestros métodos a las funciones lambdas también es necesario crear Integration Request para todas las funciones lambdas que requieran parámetros la configuración es muy sencilla, en la figura 4.8 podemos observar el Integration Request para la función lambda graph_instance.

```
#set($inputRoot = $input.path('$'))
{
   "action":"$input.params('action')"
}
```

Figura 4-8 Integration request (application/json) for graph_instance lambda

Una vez que todo este configurado podemos entrar a la sección de Dashboard y encontrar la ruta de invocación principal de nuestra API.

https://*my-api-id*.execute-api.*region-id*.amazonaws.com/graph/

5. RESULTADOS Y DISCUSIÓN

5.1. Resultados

En este capítulo se presentarán los resultados obtenidos y analizados durante la fase de experimentación de este TOG, analizaremos los resultados de nuestras API y probaremos su funcionamiento.

Cabe mencionar que para este trabajo no se logró desarrollar una función para cargar datos por cuestión de tiempo, por lo tanto, estos fueron insertados manualmente a través de SSH; En el futuro está la posibilidad de realizar esta función para aumentar las características de nuestro servicio. Otro punto para notar es que para efecto de pruebas y una mejor visualización usaremos una muestra de nuestra base de datos compuesta de 500 nodos aproximadamente.

- **graph_instance, graph_neo4j_config:**

Para probar las siguientes lambdas utilizamos POSTMAN, en primera instancia llamamos a graph/instance/start para iniciar el servidor que contiene nuestra base de datos en conjunto invocamos a graph/instance/status para verificar que nuestra base de datos esta lista (Figura 5.1).

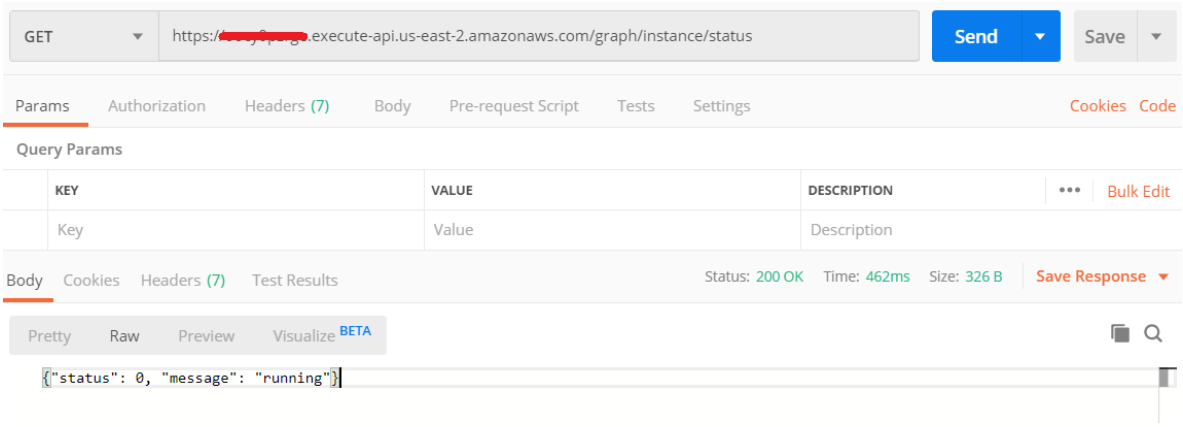

Figura 5-1 Llamada a graph_instance usando POSTMAN

Una vez que nuestra instancia se encuentra encendida invocamos a graph/config/ usando GET para obtener la lista de las bases de datos disponibles y la base de datos especificada (Figura 5.2). Una vez hecho esto procedemos a escoger la base de datos deseada usando graph/config/tweetv1 POST (Figura 5.3).

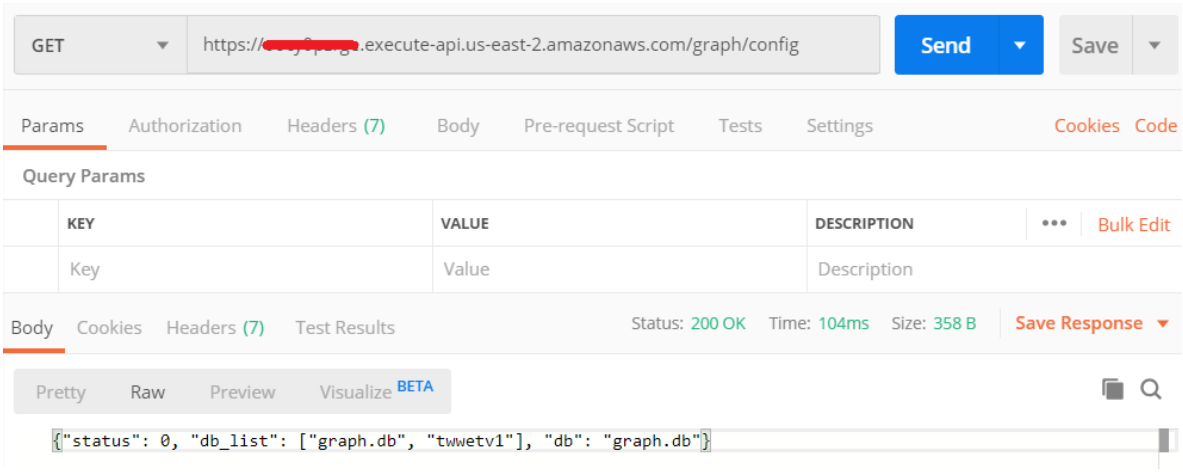

Figura 5-2 Llamada a graph_cofig para listar instancias y verificar instancia actual

Es importante mencionar que cada vez que se use este método para seleccionar una base de datos es necesario reiniciar la instancia usando graph/instance/restart.

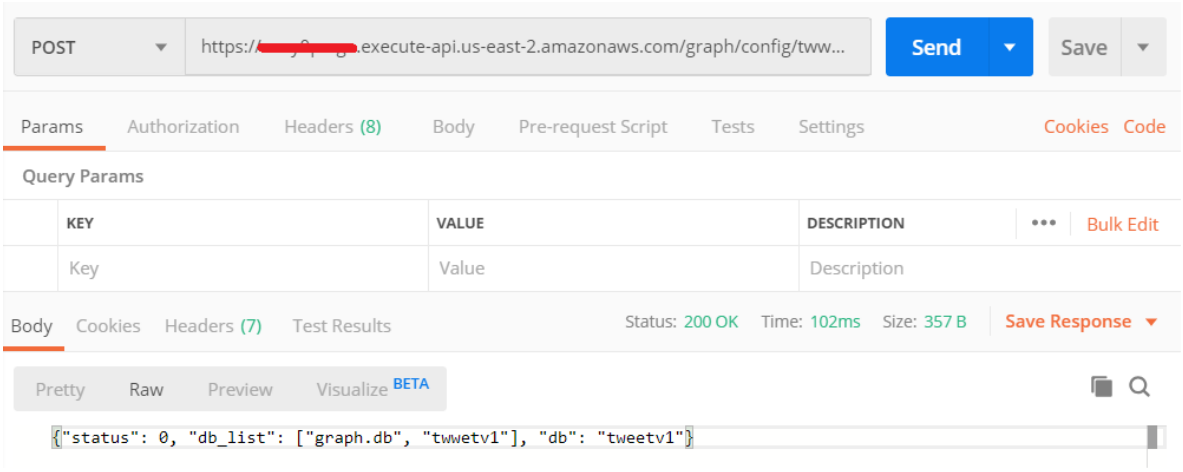

Figura 5-3 Llamada a graph_config para selección de instancia

- **graph_shema**

Una vez que tenemos nuestra base de datos seleccionada y nuestra instancia esta encendida podemos usar el resto de las funciones de la API, graph_schema es uno de los métodos principales a utilizar ya que le permitirá al usuario conocer el esquema de nuestra base de datos, y de esta manera saber qué tipos de operaciones se puede realizar. La figura 5.4 muestra el esquema de la base de datos.

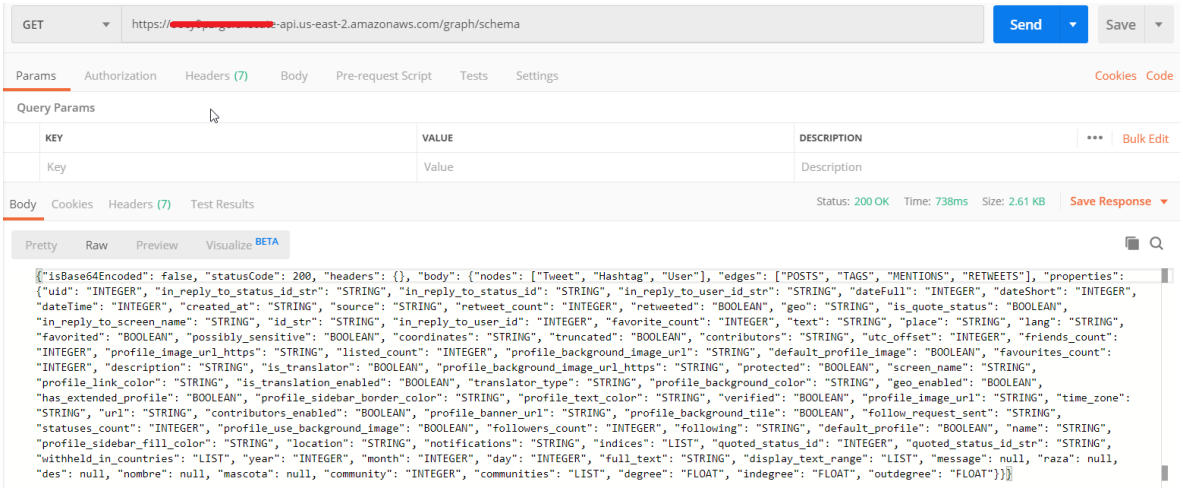

Figura 5-4 Llamada a graph_schema usando POSTMAN

- **graph_centrality, graph_louvain, graph_layout**

Las siguientes funciones nos permiten agregar atributos al grafo, para nuestro servicio contamos con 3 algoritmos, centralidad de grado, louvain y layout, al invocarlas estas agregaran nuevos atributos al grafo como podemos observar en la figura 5.5.


```
{
  "dateTime": 174905,
  "year": 2018,
  "degree": 1.0,
  "truncated": false,
  "created_at": "Sun Jan 28 23:49:05 +0000 2018",
  "dateFull": 20180128174905,
  "source": "<a href="http://twitter.com/Twitter for Android</a>",
  "community": 0,
  "retweet_count": 629,
  "retweeted": false,
  "uid": 957762484261261313,
  "dateShort": 20180128,
  "is_quote_status": false,
  "month": 1,
  "outdegree": 3.0,
  "full_text": "RT @Pajaropolitico: Cámaras …",
  "id_str": "957762484261261313",
  "display_text_range": [0,140],
  "favorite_count": 0,
  "indegree": 1.0,
  "lang": "es",
  "day": 28,
  "favorited": false,
  "communities": [0, 0, 0, 0, 0]
}
```


Figura 5-5 Esquema de un nodo Tweet con atributos añadidos

- **graph_filter**

En esta fase aplicaremos distintos filtros a nuestra base de datos con el fin de analizar nuestra base de datos y observar patrones que se presenten, a continuación, se puede observar los resultados:

Primero obtendremos un grafo sin aplicar ningún filtro, esto para tener una base visual de como luce nuestra información (Figura 5.6). En la tabla 5.1 se encuentra descrito el color que indica el tipo de nodo.

Tabla 5-1 identificación de nodos por color

Tipo de Nodo	Color
User	Amarillo
Tweet	Verde
Hashtag	Negro

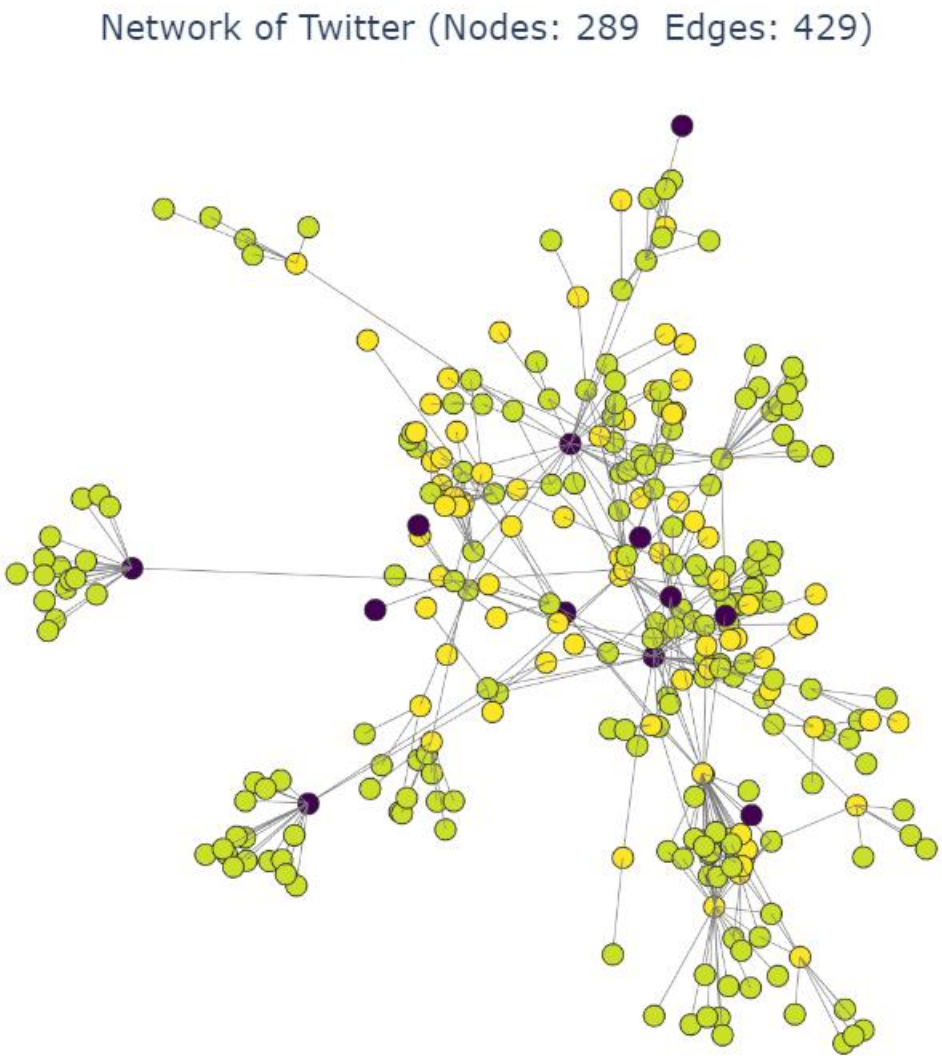

Figura 5-6 Grafo sin Filtros

En la figura 5.8 podemos observar un grafo con filtro el cual fue generado usando la siguiente configuración de filtros (figura 5.7):

```
{
   "limit":150,
   "filters":[
      {
         "element":"v",
         "element_type":"Tweet",
         "attr":"retweet_count",
         "operator":">",
         "data":100
      },
      {
         "element":"v",
         "element_type":"Tweet",
         "attr":"full_text",
         "operator":"contains",
         "data":"MarcoAntonioSanchezFlores"
      }
   ],
   "unions":[
      "or"
   ]
}
```

Figura 5-7 JSON usado graph_filter para generar imagen 5.8

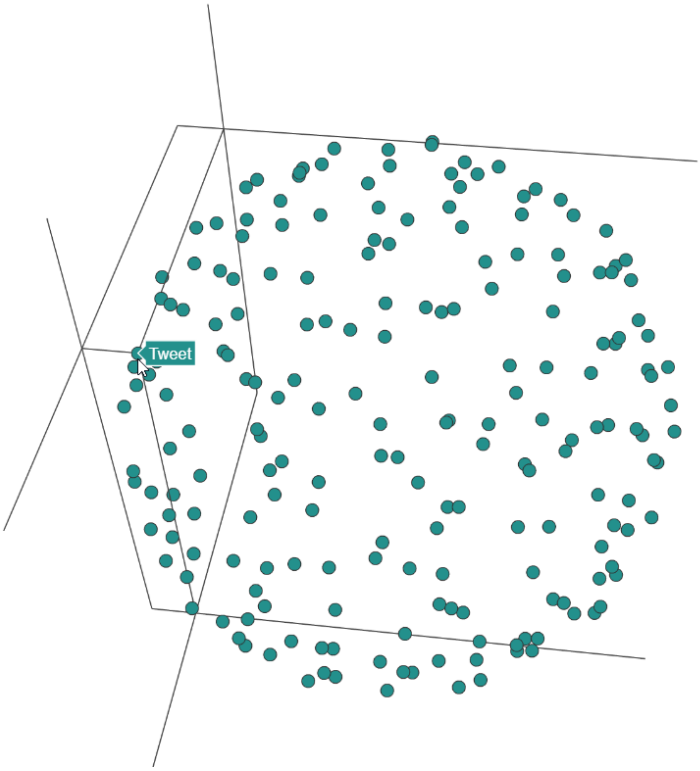

Figura 5-8 Grafo usando filtros figura 5.7

- **graph_subgraph**

En esta última fase generaremos y compararemos el grafo original con un subgrafo generado para observar relaciones indirectas, estas vistas representan información adicional y nos permiten observar una nueva perspectiva de los datos.

Para nuestra prueba generaremos un subgrafo de relaciones *User* a *User* con el fin de identificar de una manera más rápida los usuarios que tienen interacción entre ellos.

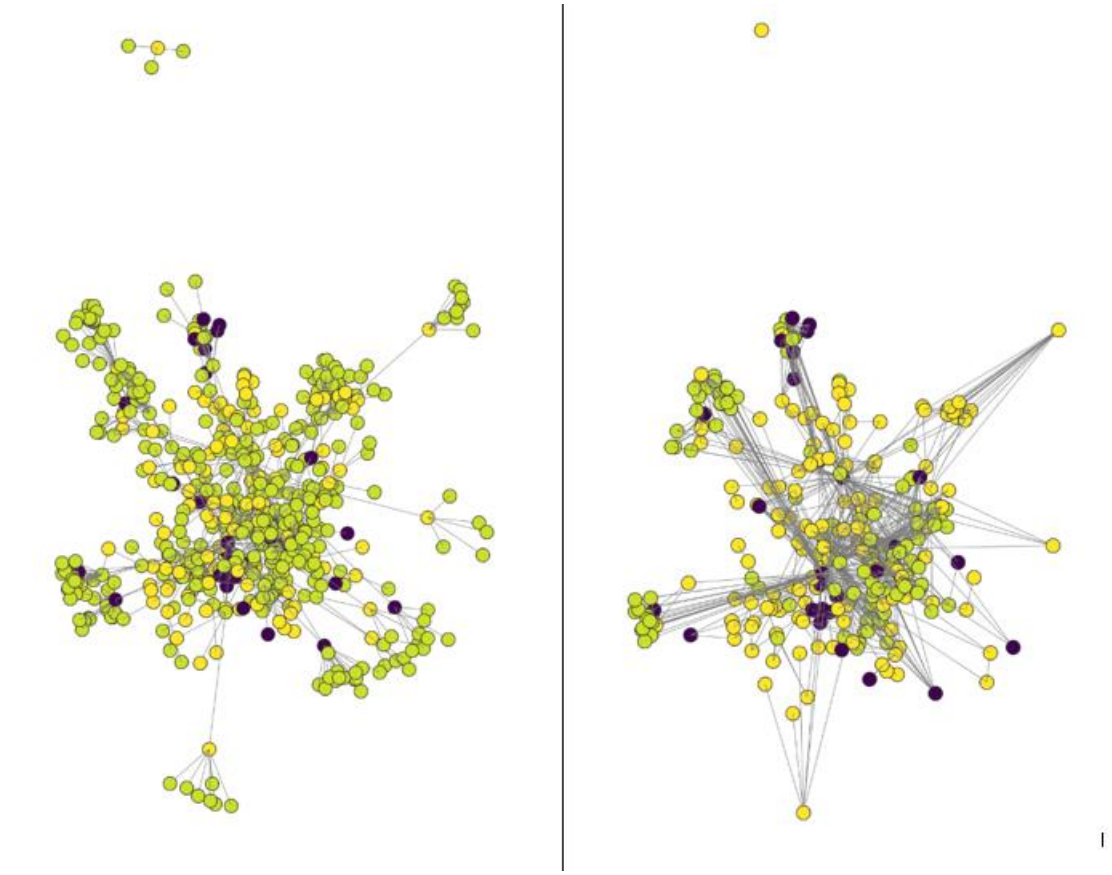

Figura 5-9 a) grafo original b) grafo con sumarización basada en nodos tipo *User*

Como observamos en la figura 5.9 el grafo original a) está compuesto de 579 nodos y 864 aristas y el mismo grafo después de la sumarización b) cambia a 290 nodos y 940 aristas, esto es un decremento notable en los nodos lo cual nos ayuda a enforcarnos más en las relaciones de usuarios con una pequeña desventaja pues incrementa el número de aristas en un 10% aproximadamente.

Aunque la visualización no es óptima se tiene la posibilidad de guardar este subgrafo y generar nuevas coordenadas contemplando solo los nodos existentes en este subgrafo.

5.2. Discusión

Como parte de la discusión observamos que dentro de los resultados obtenidos en este trabajo contamos con una capa de servicios básicos para operar y analizar un grafo. Los servicios disponibles por el momento nos permiten hacer un análisis de la información al darnos la posibilidad de hacer filtros de datos para eliminar datos innecesarios, así como un algoritmo de visualización que nos ayuda a dar sentido a la información

El análisis de grafos como servicio es una opción viable cuando se requiere de un sistema agnóstico fácil de integrar y cuando no se quiera invertir en infraestructura o recursos extras para manipular esta información. A pesar de que la API es limitada en este momento la opción de añadir nuevas funcionalidades y algoritmos es muy sencilla ya que no requiere de modificar la estructura ya existente, simplemente es cuestión de crear las funciones lambdas correspondientes y publicarlas en la API.

6. CONCLUSIONES

6.1. Conclusiones

En este trabajo se propone una nueva herramienta para análisis de grafos basada en computo en la nube y con algoritmos de análisis y visualización fue propuesta. Con los algoritmos de análisis incluidos como centralidad de grado pudimos ser capaces de identificar los nodos más influyentes de la red, así como realizar un algoritmo de visualización *force Directed* usando esta centralidad de grado. El algoritmo louvain nos ayudó a identificar comunidades, y la operación de sumarización propuesta nos ayudó a ver relaciones indirectas y así conocer que usuarios están relacionados con que usuarios sin tener que atravesar nodos que no son de interés y solo causan que la visualización sea más desordenada.

El uso del cómputo en la nube nos facilita la infraestructura para realizar todas estas operaciones sin tener que preocuparnos por requerimientos de hardware y otros recursos, además nos facilita un punto de entrada y salida para la información sencillo de utilizar y adaptable a cualquier tecnología que esté conectada a internet y requiera utilizar servicios para grafos.

Los servicios son sencillos de utilizar y no requieren de un experto que conozca de algún lenguaje de programación en específico o un lenguaje de consultas para base de datos, todo esto queda abstraído a través de la API.

Los grafos son una de las estructuras más importante para representar y manipular información, ya que ayudan a resolver problemas complejos en distintas áreas de estudio, por lo que siempre se requerirán de herramientas que ayuden con la manipulación y análisis de estas estructuras complejas, además el computo en la nube es opción viable; por lo que cada vez más compañías empiezan a cambiar su modelo de negocios a ofrecer sus soluciones como servicios web. Este proyecto plantea una propuesta base que demuestra la importancia y ventajas de proveer el análisis de los grafos como un servicio de la nube, esperando que en los futuros años sean servicios que ayuden a los diferentes sectores de la sociedad para la toma de decisiones cuando se tienen redes de información por analizar .

6.2. Trabajo Futuro

Las funciones disponibles por el momento son parte de un producto mínimo viable, pero se puede mejorar el servicio agregando más tipos de algoritmos de visualización tanto 3D como 2D, agregar más algoritmos de análisis y centralidad, así como potenciar y mejorar el uso de filtros ya que por el momento cuenta con las funciones básicas para solo 2 tipos de datos (cadenas y números), pero existe la posibilidad de incorporar tipos de datos como fechas, booleanos, etc. De la misma manera existe la posibilidad de crear nuevos operadores para estos nuevos tipos y los tipos existentes.

Otro trabajo significativo involucrando la función de filtrado seria mejorar la estructura de los parámetros de entrada para usar al máximo las capacidades del lenguaje de consulta Cypher.

Además, se propone trabajar en las lambdas y refinar el código para lograr con esto mejorar el tiempo de ejecución de nuestro servicio, esto reduciría costos y además mejoraría la experiencia de usuario al ser más responsivo.

Por otra parte, queda considerar la posibilidad de usar distintas bases de datos y definir una función para insertar los datos al sistema, así como para borrarlos o editarlos.

En resumen, el trabajo futuro es amplio y con muchas posibilidades de continuar con el desarrollo de este servicio para hacerlo más eficaz y completo y así poder integrarlo en futuras aplicaciones que requieran manipular y analizar grafos.

BIBLIOGRAFÍA

[1] C. W. a. G. Franck, "Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions," TOG Info Net Vis, Fredericton, NB, 1995.

[2] C. M. K. L. K.-L. M. Oh-Hyun Kwon, "A Study of Layout, Rendering, and Interaction Methods for Immersive Graph Visualization," *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS,* 2016.

[3] S. Bryson, "Virtual Reality in Scientific Visualization," *Comm. ACM,* vol. 39, no. 5, pp. 62-71, 1996.

[4] R. Rousseau, "Social network analysis: a powerful," *Journal of Information Science,* 2002.

[5] C. L. E. E.-Z. M. G. David Combe, "A comparative study of social network analysis tools," *WEB INTELLIGENCE & VIRTUAL ENTERPRISES,* 2010.

[6] N. Akhtar, "Social Network Analysis Tools," *Department of Computer Engineering, Zakir Husain College of Engineering & Technology,* 2016.

[7] "Overview of NetworkX," Octubre 2019. [Online]. Available: https://networkx.github.io/documentation/stable/.

[8] "igraph – The network analysis package," Noviembre 2018. [Online]. Available: https://igraph.org/.

[9] V. B. Andrej Mrvar, Exploratory Social Network Analysis with Pajek, Cambridge University Press, 2012.

[10] "The Open Graph Viz Platform," 2019. [Online]. Available: https://gephi.org/.

[11] E. N.-S. Marcin Mincera, "Application of Social Network," *Research and Academic Computer Network (NASK),* 2012.

[12] N. Kaur, "Design and Develop a Framework for Social Network Analysis," *THAPAR university,* 2016.

[13] J. A. M. k. D. C. Robert Pienta, "Scalable Graph Exploration and Visualization: Sensemaking Challenges and Opportunities," *BigComp,* 2015.

[14] V. Joshi, "A gentle Introduction to graph Theory," Marzo 2017. [Online]. Available: https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8.

[15] M. B. S. D. a. D. W. Fabian Beck, "The State of the Art in Visualizing Dynamic Graphs," *Eurographics Conference on Visualization,* p. 21, 2014.

[16] "Features," Gephi, [Online]. Available: https://gephi.org/features/.

[17] J. Y. Jonathan L. Gross, "Handbook of graph theory," in *Handbook of graph theory*, 2004.

[18] "Grafos," [Online]. Available: http://decsai.ugr.es/~jfv/ed1/c++/cdrom4/paginaWeb/grafos.htm.

[19] "The Neo4j Graph Algorithms User Guide v3.5," Neo4j, [Online]. Available: https://neo4j.com/docs/graph-algorithms/3.5/.

[20] S. Fortunato, "Community detection in graphs," *Complex Networks and Systems Lagrange Laboratory, ISI Foundation.*

[21] S. G. Kobourov, "Spring Embedders and Force Directed Graph Drawing Algorithms," *University of Arizona,* 2012.

[22] M. Grandjean, "Introduction à la visualisation de données, l'analyse de réseau en histoire," *Geschichte und Informatik ,* 2015.

[23] " Fruchterman-Reingold gephi/gephi," Febreo 2015. [Online]. Available: https://github.com/gephi/gephi/wiki/Fruchterman-Reingold.

[24] "Neo4j," [Online].

[25] "What is cloud computing?," Microsoft, [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

[26] "What is AWS," Amazon Web Servics, [Online]. Available: https://aws.amazon.com/what-is-aws/.

[27] " Que es Amazon EC2? Amazon Elastic Cloud Computing," Amazon Web Services, [Online]. Available: https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html.

[28] "AWS Lambda - Caracteristicas del Producto," [Online]. Available: https://aws.amazon.com/es/lambda/features/.

[29] "Amazon API Gateway - Amazon Web Services," Amazon Web Services, [Online]. Available: https://aws.amazon.com/es/api-gateway/.

[30] "Amazon EBS Pricing - Amazon Web Services," [Online]. Available: view-source:https://aws.amazon.com/ebs/pricing/.

[31] "AWS Marketplace: Neo4j Graph Database - Community Edition," [Online]. Available: https://aws.amazon.com/marketplace/pp/Neo4j-Neo4j-Graph-Database-Community-Edition/B071P26C9D.

[32] "AWS Lambda – Pricing," [Online]. Available: https://aws.amazon.com/lambda/pricing/.

[33] "Welcome to Paramiko! — Paramiko documentation," [Online]. Available: http://www.paramiko.org/.

[34] "Precios de Amazon API Gateway – Amazon Web Services," [Online]. Available: https://aws.amazon.com/es/api-gateway/pricing/.