



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Plataforma Big data para el almacenamiento de flujos de
información marítima*

Grado en Ingeniería Mecánica

ALUMNO: Javier Vila Aparicio
DIRECTOR: Norberto Fernández García
CURSO ACADÉMICO: 2020-2021

UniversidadeVigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Plataforma Big data para el almacenamiento de flujos de
información marítima*

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

UniversidadeVigo

RESUMEN

Actualmente, es obligatorio que gran parte de las embarcaciones utilicen el Sistema de Identificación Automática (AIS). Este sistema emite cada cierto tiempo una serie de eventos con información relevante del barco, como la identidad, la posición, el rumbo o la velocidad, para darlos a conocer a otros barcos o estaciones en tierra. Los sistemas de vigilancia y control del tráfico marítimo analizan el flujo continuo de eventos de las radiobalizas AIS, instaladas en las embarcaciones comerciales y/o de recreo, para detectar anomalías en la mar. Debido al volumen de datos, se hace necesario el uso de herramientas Big data para almacenar la información y permitir su consulta y análisis.

En este proyecto se realiza un estudio y comparación de algunas de las herramientas Big data existentes. Tras ello, se selecciona Elasticsearch por sus características favorables en este contexto, y se desarrolla el proceso de instalación y configuración. Además, se prueban las funcionalidades más útiles de cara a poder realizar consultas de potencial interés. Finalmente, se evalúa empíricamente el funcionamiento de la herramienta con datos reales extraídos a partir de 5 millones de tramas AIS proporcionadas por el Centro de Operaciones y Vigilancia de Acción Marítima (COVAM), para obtener resultados del tiempo de ejecución de las consultas y estudiar la viabilidad de la herramienta en este ámbito.

PALABRAS CLAVE

Big data, AIS, Elasticsearch, Consultas, Herramientas

AGRADECIMIENTOS

Este proyecto pone punto y final a mi camino por la Escuela Naval Militar, pero este camino no lo he realizado solo, sino acompañado de muchas personas que me han apoyado en cada paso que he dado, es por ello, que me gustaría dedicarles este apartado como agradecimiento.

En primer lugar, quería dedicar mis primeras palabras a mi familia. Ellos han sido siempre un apoyo constante e incondicional a lo largo de mi vida. Todo lo que soy, lo que he conseguido y lo que me queda por conseguir es gracias a ellos. Son mi ejemplo a seguir, enseñándome valores tan importantes como el esfuerzo, el sacrificio y la humildad.

En segundo lugar, agradecer su labor a mi tutor Norberto Fernández García, por su interés, dedicación y ayuda constantes a lo largo de estos meses de desarrollo del TFG.

Así mismo, me gustaría mostrar mi gratitud al COVAM por haber proporcionado los datos que se han usado en las pruebas.

Y por último a mi promoción, la 421-151, por acompañarme a lo largo de estos cinco años y enseñarme el verdadero valor del compañerismo y del trabajo en equipo. Especial mención merecen los más íntimos, que han hecho que el paso por esta escuela sea mucho más fácil, por sus consejos y apoyo en los momentos más duros, y por estar siempre ahí cuando se les necesitaba.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	6
1 Introducción y objetivos	7
1.1 Big data	7
1.1.1 Big data en el ámbito marítimo	7
1.1.2 Desafíos y retos del Big data marítimo.....	7
1.2 AIS	8
1.2.1 ¿Qué es AIS?	8
1.2.2 Funcionamiento del sistema AIS	8
1.3 Armada 4.0.....	10
1.3.1 Transformación digital de la Armada	10
1.3.2 Situación actual del COVAM.....	11
1.3.3 Modernización del COVAM	11
1.4 Objetivos del TFG.....	12
1.5 Estructura del TFG.....	12
2 Estado del arte	13
2.1 Análisis de las herramientas.....	13
2.1.1 MongoDB	13
2.1.2 InfluxDB.....	18
2.1.3 KsqlDB	22
2.1.4 Elasticsearch	27
2.1.5 MySQL	33
2.1.6 Apache Kylin.....	38
2.1.7 Apache Cassandra.....	43
2.1.8 Redis	49
2.1.9 Comparación de las herramientas	53
3 Desarrollo del TFG.....	59
3.1 Proceso de instalación Elasticsearch.....	59
3.1.1 Características del equipo.	59
3.1.2 Proceso de instalación de Elasticsearch.....	59
3.1.3 Funcionalidades básicas.....	61
3.1.4 Flujos de datos	71

3.1.5 Consultas geográficas	75
4 Prueba.....	87
4.1.1 Fichero de tramas AIS	87
4.1.2 Tipos de consultas.....	90
4.1.3 Análisis de las consultas	92
4.1.4 Resumen de los resultados.....	103
5 Conclusiones y líneas futuras	105
5.1 Conclusiones	105
5.2 Líneas futuras.....	106
6 Bibliografía.....	107
Anexo I: Resultados obtenidos.....	117

ÍNDICE DE FIGURAS

Figura 1-1 Mapa de la situación marítima producido por MarineTraffic donde se puede ver el posicionamiento en directo de multitud de buques de todo tipo [1]	8
Figura 1-2 Sala de operaciones del COVAM [140]	11
Figura 2-1 Logotipo MongoDB [127]	13
Figura 2-2 Gráfica DBSM de datos documentales según su popularidad [126]	14
Figura 2-3 Ejemplo lenguaje de consulta de MongoDB [102]	15
Figura 2-4 Arquitectura MongoDB [8]	15
Figura 2-5 Gráfico de la arquitectura MongoDB [9]	16
Figura 2-6 Curva de descargas del popular juego de móvil Crazy Taxi: City Rush [10]	17
Figura 2-7 Logotipo InfluxDB [123]	18
Figura 2-8 Ejemplo del lenguaje de programación Flux [103]	18
Figura 2-9 Gráfica de los DBMS de datos temporales según su popularidad [12]	19
Figura 2-10 Ejemplo flujo de datos en InfluxDB [11]	20
Figura 2-11 Stack InfluxDB [13]	20
Figura 2-12 Arquitectura técnica usando InfluxDB Enterprise y Telegraf Agents [15]	22
Figura 2-13 Logotipo KsqlDB [122]	23
Figura 2-14 Ejemplo del lenguaje de consulta de KsqlDB [77]	23
Figura 2-15 Arquitectura básica Apache Kafka [18]	24
Figura 2-16 Arquitectura procesamiento de flujo común [19]	25
Figura 2-17 Arquitectura por bloques [19]	25
Figura 2-18 Arquitectura con Kafka y KsqlDB [19]	26
Figura 2-19 Arquitectura interna [19]	26
Figura 2-20 Logotipo Elasticsearch [138]	28
Figura 2-21 Gráfica de bases de datos de búsqueda según su popularidad [25]	29
Figura 2-22 Ejemplo del lenguaje de consulta de Elasticsearch [135]	30
Figura 2-23 Índice dividido en dos <i>shards</i> (P0 y P1), 3 nodos y 2 réplicas por <i>shard</i> . [23]	31
Figura 2-24 Fallo de un nodo [23]	31
Figura 2-25 Logotipo MySQL [133]	33
Figura 2-26 Ejemplo del lenguaje de MySQL [79]	33
Figura 2-27 Gráfica DBMS de datos relacionales según su popularidad [31]	34
Figura 2-28 Arquitectura lógica de MySQL [33]	35
Figura 2-29 Arquitectura Netflix con Oracle [34]	37
Figura 2-30 Arquitectura Netflix con Cassandra y MySQL [34]	37

Figura 2-31 Logotipo Apache Kylin [35].....	38
Figura 2-32 Gráfica de las bases de datos relacionales según su popularidad	39
Figura 2-33 Ejemplo de cubo OLAP [124]	40
Figura 2-34 Esquema visión general arquitectura Apache Kylin [37]	40
Figura 2-35 Ejemplo cubo OLAP en Apache Kylin [119]	41
Figura 2-36 Arquitectura Apache Kylin [35] [38]	42
Figura 2-37 Ejemplo del lenguaje de consulta de Apache Kylin [59]	42
Figura 2-38 Logotipo Cassandra [132]	43
Figura 2-39 Ejemplo del lenguaje CQL de Apache Cassandra [136]	44
Figura 2-40 Gráfica de los DBMS de columna ancha según su popularidad.....	45
Figura 2-41 Teorema CAP [40].....	45
Figura 2-42 Resumen del proceso de escritura [42]	47
Figura 2-43 Ejemplo estructuración de nodos Cassandra [129].....	47
Figura 2-44 Logotipo Redis [134].....	49
Figura 2-45 Gráfica de los DBSM clave-valor según su popularidad	50
Figura 2-46 Ejemplo del lenguaje de consulta de Redis [137].....	50
Figura 2-47 Ejemplo de <i>cluster</i> con replicación asíncrona [139]	51
Figura 3-1 Contenido de la carpeta Elasticsearch-7.10.2	60
Figura 3-2 Test de funcionamiento de Elasticsearch.....	60
Figura 3-3 Test de funcionamiento del <i>cluster</i> de Elasticsearch mediante <i>cat health API</i>	61
Figura 3-4 Creación de un índice	61
Figura 3-5 Borrar un índice	62
Figura 3-6 Solicitud PUT	62
Figura 3-7 Solicitud GET	63
Figura 3-8 Ejemplo de cuentas bancarias del fichero <i>accounts.json</i>	63
Figura 3-9 Documentos indexados.....	63
Figura 3-10 Solicitud de búsqueda en el fichero <i>accounts.json</i>	64
Figura 3-11 Solicitud de búsqueda de una dirección en concreto	65
Figura 3-12 Solicitud de búsqueda combinando varios criterios	66
Figura 3-13 Solicitud de búsqueda con filtro de rango	67
Figura 3-14 Búsqueda de prefijo	67
Figura 3-15 Restricción de campos "fields" en la consulta	68
Figura 3-16 Uso de "sort" para ordenar los resultados de la búsqueda.....	68
Figura 3-17 Solicitud de búsqueda con agregación.....	69
Figura 3-18 Solicitud de búsqueda con combinación de agregaciones.....	70
Figura 3-19 Índices de respaldo	71

Figura 3-20 Solicitud de lectura enviada a los índices de respaldo.....	72
Figura 3-21 Índice de escritura.....	72
Figura 3-22 Creación de política de ciclo de vida.....	73
Figura 3-23 Creación plantilla de índice (<i>index template</i>).....	74
Figura 3-24 Creación de un flujo de datos	74
Figura 3-25 Comprobación del estado del ciclo de vida	75
Figura 3-26 Consulta <i>geo_point</i> y <i>geo_shape</i>	76
Figura 3-27 Continuación consulta <i>geo_point</i> y <i>geo_shape</i>	77
Figura 3-28 Consulta <i>geo_bounding_box</i> y definición del <i>box</i>	78
Figura 3-29 Consulta para recuperar <i>geo_point</i> y <i>geo_shape</i>	78
Figura 3-30 Consulta <i>geo_distance</i>	79
Figura 3-31 Consulta para recuperar <i>geo_point</i> y <i>geo_shape</i>	79
Figura 3-32 Consulta <i>geo_polygon</i>	80
Figura 3-33 Definición <i>mapping geo_shape</i> e introducción de datos.....	81
Figura 3-34 Consulta <i>geo_shape</i>	82
Figura 3-35 Consulta <i>pre-indexed shape</i>	83
Figura 3-36 Consulta recuperación de <i>geo_shape</i>	84
Figura 4-1 Acceso al servidor	90
Figura 4-2 Diagrama de barras consultas ALL	93
Figura 4-3 Diagrama de barras consultas PORT.....	94
Figura 4-4 Diagrama de barras consultas LENGTH.....	95
Figura 4-5 Diagrama de barras consultas POS.....	96
Figura 4-6 Diagrama de barras consulta CELL.....	97
Figura 4-7 Diagrama de barras consultas TIME	98
Figura 4-8 Diagrama de barras consultas MMSI	99
Figura 4-9 Diagrama de barras consultas NEAR.....	100
Figura 4-10 Diagrama de barras consultas ZONE	101
Figura 4-11 Diagrama de barras consultas NEARGEO.....	103

ÍNDICE DE TABLAS

Tabla 1 Comparación funcionalidad AIS clase A y B [3].....	9
Tabla 2 Comparación del periodo de transmisión [3]	9
Tabla 3 Comparación de los datos transmitidos [3]	10
Tabla 4 Tabla comparativa de las herramientas	55
Tabla 5 Comparación rendimiento InfluxDB y Elasticsearch.....	56
Tabla 6 Estadísticas consultas ALL	92
Tabla 7 Estadísticas consultas PORT	93
Tabla 8 Estadísticas consultas LENGTH	94
Tabla 9 Estadísticas consultas POS.....	96
Tabla 10 Estadísticas consulta CELL.....	97
Tabla 11 Estadísticas consultas TIME	98
Tabla 12 Estadísticas consultas MMSI	99
Tabla 13 Estadísticas consultas NEAR	100
Tabla 14 Estadísticas consultas ZONE	101
Tabla 15 Estadísticas consultas NEARGEO	102
Tabla 16 Resumen de los resultados	103
Tabla 17 Resultados consultas ALL.....	117
Tabla 18 Resultados consultas PORT	118
Tabla 19 Resultados consultas LENGTH	119
Tabla 20 Resultados consultas POS	120
Tabla 21 Resultados consultas CELL	121
Tabla 22 Resultados consultas TIME.....	122
Tabla 23 Resultados consultas MMSI.....	123
Tabla 24 Resultados consultas NEAR.....	124
Tabla 25 Resultados consultas ZONE.....	125
Tabla 26 Resultados consultas NEARGEO	126

1 INTRODUCCIÓN Y OBJETIVOS

Antes de comenzar con este proyecto es necesario conocer dos conceptos: el Big Data y el sistema AIS. Estas dos herramientas están indirectamente relacionadas, ya que el AIS es un sistema de ayuda a la navegación marítima que maneja una gran cantidad de datos relativos al entorno marítimo y el Big Data es una tecnología para gestionar de manera eficiente esa cantidad extraordinaria de datos que, sería realmente difícil o incluso imposible de hacer con las herramientas tradicionales. Se hablará también de la importancia de este marco tecnológico de cara a la evolución hacia la denominada Armada 4.0 y su impacto en el COVAM.

1.1 Big data

1.1.1 Big data en el ámbito marítimo

La cantidad de datos en el dominio marítimo está creciendo de manera exponencial sin precedentes. Cada mes se recopilan terabytes de datos procedentes de sensores, boyas, embarcaciones y satélites. En ocasiones, se está llegando incluso a petabytes de datos.

De la mano de la cuarta revolución industrial, vienen las tecnologías de Big data, las cuales se están aplicando actualmente a un número creciente de dominios y disciplinas. Sin embargo, en el ámbito marítimo esta tecnología está muy poco desarrollada actualmente.

Se están volviendo cada vez más comunes los sensores en todo tipo de embarcaciones y buques. Además del GPS, utilizado para el posicionamiento y una navegación segura, se están incorporando hoy en día otros sensores que proporcionan flujos de datos sobre el rendimiento del barco, su condición marinera y de carga, temperatura y humedad. Además de los sensores incorporados en los buques, también hay otros desplegados en la mar en forma de boyas o incluso sumergidos para obtener datos de la temperatura, corrientes, viento, concentración de sal, etc., lo que añade muchos más datos a los ya vigentes, haciendo que el uso de la tecnología Big data sea necesario [1].

1.1.2 Desafíos y retos del Big data marítimo

Tecnológicamente hablando, el Big data tiene que cumplir cinco aspectos que se conocen como las 5V, estos son: volumen, ya que la generación y almacenamiento de datos es cada vez mayor; variedad, que indica heterogeneidad en los datos; velocidad, los datos cada vez se generan más rápidamente; valor, nos dicta la utilidad de los datos; y por último la veracidad, referida a posibles problemas de ruido y calidad de los datos. Por tanto, es necesario tener sistemas que sean capaces de integrar y analizar datos heterogéneos procedentes de fuentes diversas. Además, estos sistemas deben estar conectados a la nube para garantizar el acceso a los flujos de datos en tiempo real.

Según lo establecido en el convenio SOLAS (*Safety of Life at Sea*), ciertos buques, están obligados en todos sus tránsitos a recopilar y almacenar una gran cantidad de datos de su entorno, como: posición, rumbo, velocidad, radiocomunicaciones terrestres y espaciales y las condiciones meteorológicas. Hay organizaciones científicas y gubernamentales como NOAA (*National Oceanic and Atmospheric Administration*), NASA (*National Aeronautics and Space Administration*) y NODC (*National Oceanographic Data Center*), que se encargan de recopilar todos los datos de la nube ofrecidos por los sensores de los buques y las boyas, para poner estos datos a disposición del público en infraestructuras Web, de tal manera, que las compañías marinas puedan ofrecer servicios de mapas en vivo (véase Figura 1-1) e información actualizada sobre cada buque que se encuentre en la mar [1].

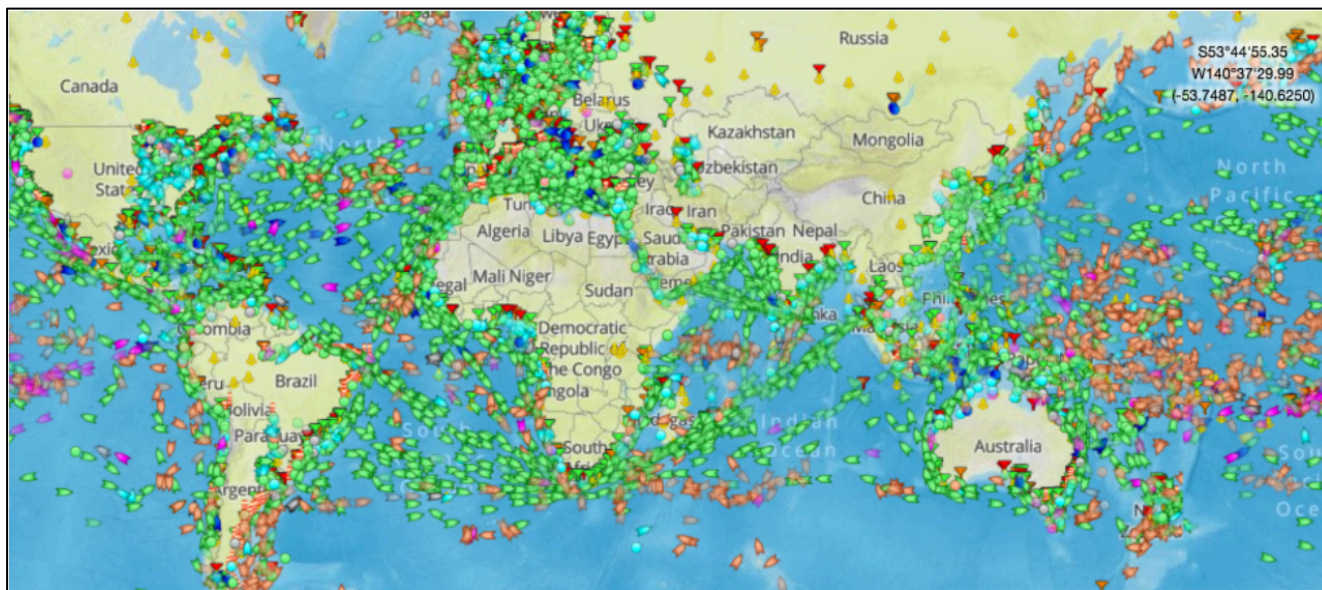


Figura 1-1 Mapa de la situación marítima producido por MarineTraffic donde se puede ver el posicionamiento en directo de multitud de buques de todo tipo [1]

Resumiendo, en líneas generales podemos decir que Big data consiste en datos cuyo volumen, velocidad de adquisición, variedad y valor potencial superan la capacidad de los sistemas tradicionales de gestión de datos.

1.2 AIS

1.2.1 ¿Qué es AIS?

AIS se corresponde con las siglas de *Automatic Identification System*. Es un sistema de seguimiento automático que muestra las embarcaciones que se encuentran a nuestro alrededor. El objetivo principal del sistema AIS, es permitir a los buques comunicar su posición y otro tipo de información relevante, para evitar así, situaciones que puedan conllevar un peligro para la navegación.

Cabe reseñar que es solo una ayuda para la navegación, no debe usarse como medio para evitar colisiones. Las Estaciones de Tráfico utilizan este sistema para identificar, localizar y monitorear las embarcaciones de su entorno.

La Organización Marítima Internacional declara por medio del convenio SOLAS que, todos los buques con un desplazamiento mayor a 300 toneladas, los que realicen tránsitos internacionales y los buques de transporte deben llevar el sistema AIS instalado de manera obligatoria [2].

1.2.2 Funcionamiento del sistema AIS

Este sistema usa dos frecuencias, y está sujeto a la misma limitación de alcance que el VHF, es decir, la línea de visión (entre 20 y 30 millas náuticas). También existe AIS vía satélite, sin embargo, no es tan

efectivo como el terrestre, ya que el esquema de acceso por radio TDMA (*Time Division Multiple Access*) definido en el estándar AIS, crea 4.500 franjas horarias disponibles cada minuto, pero este, puede verse abrumado por las grandes huellas de recepción de satélites y el creciente número de transceptores AIS. Lo que resulta en colisiones de mensajes, que el receptor satélite no puede procesar. Es por ello que, a día de hoy no es muy efectivo el sistema AIS vía satélite.

Actualmente, existen dos clases de AIS [3]:

- Clase A: Obligatorio para los buques sujetos al convenio SOLAS.
- Clase B: Para embarcaciones de recreo, siendo optativo su uso.

En las siguientes tablas se podrá ver con mayor facilidad la diferencia entre las dos clases de AIS y todo lo que nos puede proporcionar esta herramienta:

FUNCIÓN	CLASE A	CLASE B
Potencia de transmisión	12,5 W	2 W
Periodo de transmisión	Hasta cada 2-3 segundos	Hasta cada 30 segundos
Datos del trayecto	SÍ	NO
Asignación de intervalo de tiempo	SÍ	NO
Conexión GPS Externa	SÍ	SÍ

Tabla 1 Comparación funcionalidad AIS clase A y B [3]

DATOS EN MOVIMIENTO	CLASE A	CLASE B
Barco fondeado	3 minutos	3 minutos
SOG 0-2 nudos	10 segundos	3 minutos
SOG 2-14 nudos	10 segundos	30 segundos
SOG 14-23 nudos	6 segundos	30 segundos
SOG >23 nudos	2 segundos	30 segundos
Datos estáticos	6 minutos	6 minutos

Tabla 2 Comparación del periodo de transmisión [3]

INFORMACIÓN TRANSMITIDA	CLASE A	CLASE B
MMSI (<i>Maritime Mobile Service Identity</i>) + Nombre del Barco + <i>Call Sign</i>	SÍ	SÍ
Posición + COG + SOG	SÍ	SÍ
Dirección	SÍ	SÍ
Tasa de giro	SÍ	NO
Estado de navegación	SÍ	NO
Número IMO (<i>International Maritime Organization</i>)	SÍ	NO
Modelo de barco	SÍ	SÍ
Dimensiones	SÍ	SÍ

Tabla 3 Comparación de los datos transmitidos [3]

El sistema tiene incorporado un transpondedor, el cual va transmitiendo por radiodifusión la siguiente información [3]:

- Posición GPS, SOG (*Speed over ground*), COG (*Course over ground*).
- Rumbo y velocidad de giro.
- Número MMSI (*Maritime Mobile Service Identity*).

Los datos estáticos son transmitidos en intervalos de 6 minutos, un tiempo de refresco mayor que en los datos anteriores, donde estaríamos hablando del orden de segundos ya que son datos dinámicos y más relevantes para la seguridad en la navegación. Los datos estáticos son los siguientes: [3]

- Nombre del buque, distintivo de llamada y MMSI.
- Dimensiones y tipo de buque.
- Datos del tránsito (Destino / ETA).
- Estado de Navegación (Si se encuentra fondeado o navegando).
- IMO Number (Número de la Organización Marítima Internacional).

1.3 Armada 4.0

1.3.1 Transformación digital de la Armada

Hoy en día, todas las organizaciones públicas y privadas, civiles y militares, se ven envueltas en lo que se conoce como la cuarta revolución industrial, caracterizada por la intensiva utilización de nuevas tecnologías digitales, y por el cambio tecnológico, cada vez más acelerado.

Es por ello, que resulta esencial para la Armada, el desarrollo y puesta en práctica de una transformación digital, que permita explotar al máximo todas las ventajas competitivas que nos ofrecen estas nuevas tecnologías. Esta adaptación resulta esencial para alcanzar y mantener una superioridad sobre potenciales adversarios.

Para ello, la Armada se ha apoyado en el CESTIC (Centro de Sistemas y Tecnologías de Información y Comunicaciones de la Secretaría de Estado de Defensa) para iniciar todo el proceso de transformación

digital que permita una rápida adaptación al entorno cambiante, y así, maximizar la eficiencia de los recursos materiales, financieros y humanos [4].

1.3.2 Situación actual del COVAM

Dentro de las misiones de la Armada, se encuentran las operaciones de Vigilancia y Seguridad Marítima en los espacios marítimos de soberanía e interés nacional. Para ello, es necesario tener en todo momento, un adecuado conocimiento del entorno en las diferentes áreas de operaciones marítimas. Actualmente, esta responsabilidad recae sobre el Centro de Operaciones y Vigilancia de Acción Marítima (COVAM). En la Figura 1-2, se puede ver la sala de operaciones del COVAM donde se estudia y analiza el entorno marítimo.



Figura 1-2 Sala de operaciones del COVAM [140]

En octubre de 2006, el COVAM inició el desarrollo del Sistema de Información y Vigilancia Integrado para el Conocimiento del Entorno Marítimo de la Armada (SIVICEMAR), ante la necesidad de proporcionar el conocimiento del entorno marítimo a las unidades desplegadas en operaciones de vigilancia y seguridad marítima.

Sin embargo, el SIVICEMAR no puede considerarse como un sistema, sino como un conjunto de herramientas independientes gestionadas de manera coordinada, las cuales emplean los datos proporcionados por el sistema AIS a nivel mundial. A pesar de ello, tiene la gran desventaja de que no dispone de información procedente de otros sensores (radar, ópticos...) o de posición en tiempo real de unidades propias, ni de las herramientas necesarias para su recepción, fusión, análisis y diseminación [5].

1.3.3 Modernización del COVAM

Las diferentes opciones de solución operativa pasan por proceder a la modernización de los sistemas ya existentes en el COVAM. El objetivo, es disponer de un sistema servidor de tratamiento, presentación y almacenamiento, el cual permita recibir, almacenar, procesar y combinar toda la información marítima procedente de las diversas fuentes disponibles y que posibilite su tratamiento y análisis, en entornos de área local de diferente nivel de clasificación de seguridad, mediante protocolos estandarizados, compatibles con los implementados en la UE y OTAN.

El COVAM pretende evolucionar, pasando de utilizar un sistema basado en un conjunto de herramientas y sistemas independientes, a un sistema de sistemas que permita el adecuado manejo de la información.

Para llevar a cabo esta solución, es necesario el desarrollo de un software (principalmente para la herramienta de fusión y tratamiento de la información de manera automática) y también, un hardware a medida [5].

Es por ello, que se hace necesario el uso de la tecnología Big data, para poder gestionar de manera eficiente todos los datos que maneja el COVAM diariamente. Recientemente, han cerrado contrato con la consultora informática INETUM [6] para el desarrollo de este nuevo software.

1.4 Objetivos del TFG

Como se observa en el apartado 1.3, nos encontramos en un entorno cambiante tecnológicamente hablando, es por ello que la Armada necesita adaptarse para mantener la máxima eficiencia de sus recursos y supremacía.

En la actualidad, existen numerosas herramientas Big data de código libre para el almacenamiento de flujos de información. En este proyecto, estudiaremos algunas de ellas, comparando de manera cualitativa sus ventajas e inconvenientes. Posteriormente, se seleccionará una de las herramientas estudiadas, y se procederá a la instalación, configuración y prueba en un entorno con datos AIS reales, para evaluar su rendimiento en este contexto.

1.5 Estructura del TFG

El proyecto se estructurará en cinco capítulos:

- Capítulo 1: Introducción y objetivos. En este capítulo se pretende contextualizar el proyecto, explicando brevemente qué es la tecnología Big data y el sistema AIS. Además, también se trata la situación actual del COVAM y su evolución para adaptarse a la cuarta revolución industrial.
- Capítulo 2: Estado del arte. Se describirán y compararán de forma cualitativa algunas de las herramientas de código libre que puedan ser útiles para el almacenamiento de flujos de información consistentes en tramas AIS.
- Capítulo 3: Desarrollo. De las herramientas analizadas en el Capítulo 2, se seleccionará una de ellas, y se explicará el proceso de instalación y configuración para probarla con datos reales.
- Capítulo 4: Prueba y validación. La herramienta seleccionada, se ejecutará y probará con datos AIS reales, para evaluar su funcionamiento y obtener resultados.
- Capítulo 5: Conclusiones y líneas futuras. Se describen una serie de conclusiones obtenidas en función del desarrollo del trabajo, y se estudian brevemente futuras líneas de desarrollo del trabajo realizado.

2 ESTADO DEL ARTE

En este capítulo, se realiza un análisis de algunas de las herramientas de código libre utilizadas hoy en día para el almacenamiento de datos. Posteriormente, se comparan de manera cualitativa, y se elegirá la más apropiada para el almacenamiento de datos AIS reales.

2.1 Análisis de las herramientas

En este apartado, se va a proceder a analizar ocho herramientas de código libre, para el almacenamiento de información. Las aplicaciones a analizar serán las siguientes: MongoDB, InfluxDB, KsqlDB, Elasticsearch, MySQL, Apache Kylin, Apache Cassandra y Redis.

2.1.1 MongoDB

2.1.1.1 ¿Qué es MongoDB?

MongoDB es un sistema de base de datos NoSQL, orientado a documentos, de código abierto y escrito en C++. Se caracteriza, porque, en lugar de guardar los datos en tablas, lo hace en estructuras de datos BSON (*Binary JavaScript Object Notation*) con un esquema dinámico.

Ahora bien, ¿Qué es BSON?, es un formato de intercambio de datos, usado principalmente para su almacenamiento y transferencia en la base de datos de MongoDB. Es una representación binaria de estructuras de datos y mapas. BSON significa *Binary JSON*, al ser binario es más compacto, ocupa menos, y tarda menos tiempo en transmitirse por la red, lo que hace al formato BSON mucho más eficiente que el formato JSON.

Es un proyecto de código abierto, con distribuciones disponibles para sistemas operativos Windows, GNU/Linux, OS X y Solaris. Además, es usado en multitud de proyectos o implementaciones en grandes empresas tales como, MTV Network, Craigslist, BCI o Foursquare [7].



Figura 2-1 Logotipo MongoDB [127]

Una de las principales características de MongoDB, cuyo logo puede verse en la Figura 2-1, es su velocidad, ofreciendo un balance entre rendimiento y funcionalidad gracias a su sistema de consulta de

contenidos. Además de su rapidez, cuenta con otras muchas características que se reseñan a continuación [7]:

- Consultas ad hoc: Con MongoDB se pueden realizar todo tipo de consultas. Es posible hacer búsquedas de campos, consultas de rangos y expresiones regulares.
- Indexación: El concepto de índices en MongoDB es similar al empleado en bases de datos relacionales, con la diferencia de que cualquier campo documentado puede ser indexado y se pueden añadir múltiples índices secundarios.
- Replicación: Es un proceso básico en la gestión de bases de datos. MongoDB utiliza el tipo de replicación primario-secundario, de tal manera que mientras que realizamos consultas con el primario, los secundarios actúan como réplica de datos sólo para lectura a modo de copia de seguridad. Además, estos nodos secundarios pueden elegir un nuevo nodo primario en caso de que el primario actual se vea afectado.
- Balanceo de carga: MongoDB tiene capacidad para escalar la carga trabajo. Es capaz de ejecutarse de manera simultánea en múltiples servidores, mejorando así la escalabilidad.
- Almacenamiento de archivos: MongoDB se puede utilizar también como un sistema de archivos aprovechando la capacidad de balanceo de carga y recopilación de datos.

Como se puede ver en la Figura 2-2, actualmente MongoDB se encuentra entre las herramientas de gestión de base de datos más utilizada por los usuarios. En el sector de las bases de datos documentales se encuentran en primer lugar.

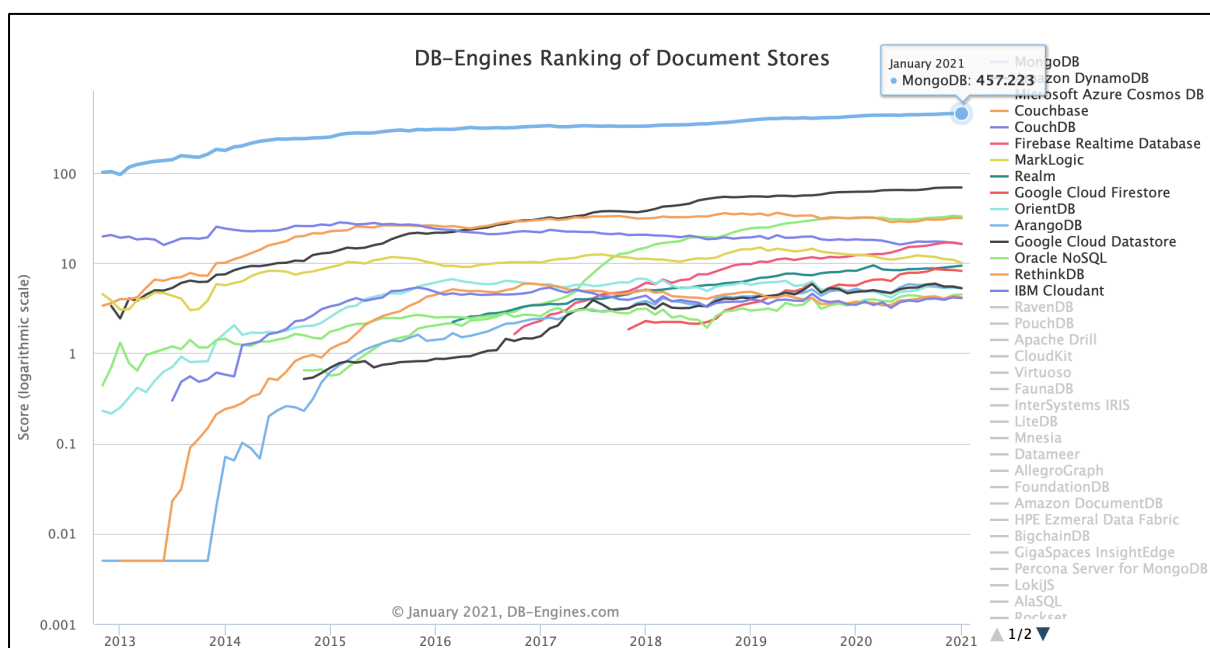


Figura 2-2 Gráfica DBSM de datos documentales según su popularidad [126]

2.1.1.2 Funcionamiento de MongoDB

MongoDB está formado por una serie de capas, las cuales se detallan a continuación [8]:

- Lenguaje de consulta de MongoDB (MQL): Es el lenguaje utilizado para trabajar con MongoDB. Esta capa es la encargada de gestionar todas las operaciones CRUD (*Create, Read, Update and Delete*) que se efectúan sobre la base de datos. En esta capa, se traducen las peticiones efectuadas desde el *driver* de la aplicación cliente mediante mensajes BSON en operaciones MongoDB. En la Figura 2-3, se puede observar un ejemplo del lenguaje de consulta de MongoDB en el que se realiza una agregación.

```
db.orders.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Figura 2-3 Ejemplo lenguaje de consulta de MongoDB [102]

- Modelo de datos de documentos MongoDB: Su función es aplicar las instrucciones entrantes de la capa anterior sobre las estructuras de datos manejadas por MongoDB. Manejando las estructuras de datos, espacios de nombres e índices y, realizando además, el replicado.
- Capa de almacenamiento: Encargada de la persistencia de datos en disco. También maneja cuestiones como: las llamadas al sistema, los accesos a disco, los sistemas de compresión y las estructuras de archivos.

Además de las capas principales, MongoDB dispone de unas capas encargadas de la seguridad y la administración. En la Figura 2-4 se puede ver cómo se relacionan entre sí las diferentes capas de la herramienta.

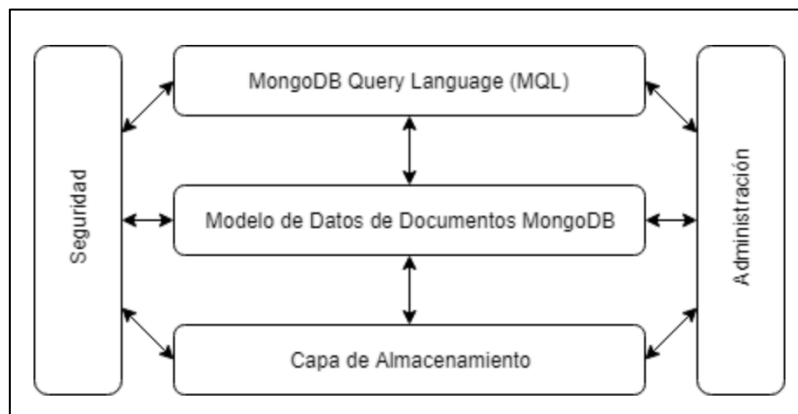


Figura 2-4 Arquitectura MongoDB [8]

MongoDB permite la creación de *clusters* de servidores para facilitar soluciones de alta disponibilidad. Existe un servidor principal, denominado nodo primario, encargado de procesar las peticiones de entrada y salida, junto con varios secundarios que actúan como réplicas de información. En el momento que un nodo principal falla, MongoDB activa el protocolo *Raft*, mediante el cual se convierte uno de los nodos secundarios en principal, para evitar así una caída del servicio.

Esta herramienta cumple con la función de ser una base de datos escalable, utilizando el concepto de *shard* (fragmento), dividiendo la base de datos en varios nodos *shard*, y añadiendo nodos en función de la cantidad de datos. Junto con el componente *Mongos*, que se encarga del enrutamiento de la información hacia los nodos *shard* implicados en la operación. [8]:

Es posible compartir datos entre los *shards* en MongoDB de manera totalmente automática, lo que reduce la tasa de fallos y, hace de esta herramienta, un sistema eficaz para la gestión de datos, altamente escalable [9].

En la Figura 2-5 se puede ver un gráfico de la arquitectura de MongoDB, donde se muestran los servidores de enrutamiento *Mongos* en color verde, los servidores de configuración en amarillo y los *shards* en azul.

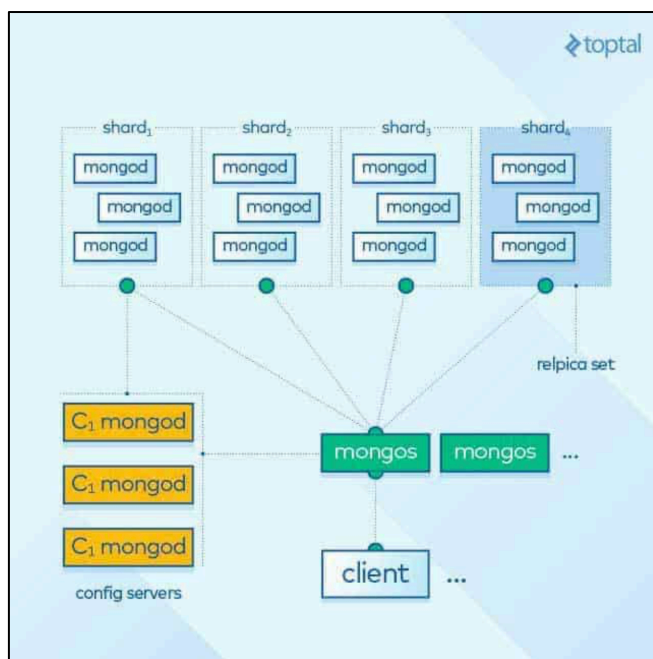


Figura 2-5 Gráfico de la arquitectura MongoDB [9]

2.1.1.3 Escenario de uso

Realizando un análisis del uso de MongoDB, obtenemos los siguientes resultados:

- Resultados en Google al buscar “MongoDB”: 24.000.000
- Seguidores en la red social Twitter (@MongoDB): 356.759 seguidores
- Fecha de búsqueda: 20 de enero de 2021

La herramienta de gestión de bases de datos documentales, MongoDB, es muy utilizada en la actualidad por grandes empresas de múltiples sectores, tales como, SEGA, EA, Google, entre otras muchas.

SEGA en particular, decidió migrar a MongoDB para simplificar las operaciones o problemas que pudieran surgir, y mejorar la experiencia recreativa de millones de jugadores.

Fue en el verano de 1991 cuando esta empresa desarrolló su primer videojuego. Los juegos han recorrido un largo camino, sin embargo, la infraestructura subyacente que impulsó estos juegos no ha cambiado mucho, basándose en bases de datos relacionales. Pero con la necesidad cada vez mayor de conseguir una gran escalabilidad, flexibilidad y creatividad, SEGA decidió optar por el servicio de MongoDB alojado en la nube, MongoDB Atlas, para brindar una mejor experiencia de juego a los usuarios.

SEGA HARDlight es un estudio de desarrollo de juegos para móvil perteneciente a SEGA. Las versiones anteriores de juegos como Sonic Jump y Sonic Dash, los cuales no hace mucho tiempo que se desarrollaron, no requerían de una conexión a Internet y no tenían funcionalidad de servidor. Estos juegos eran relativamente estáticos, ya que los desarrolladores inicialmente respaldaron los lanzamientos con una pila tecnológica interna basada en Java y MySQL, alojada en el propio centro de datos de SEGA HARDlight [10].

El procedimiento para lanzar estos juegos, implicaba probar la carga de los servidores hasta el punto de romperse, para luego aprovisionar los recursos necesarios obteniendo un punto de fallo aceptable. Esta funcionalidad era muy limitada, y podía causar interrupciones del servicio llegando al punto de fallo de los recursos aprovisionados. A medida que los juegos fueron evolucionando, comenzaron a agregar más funcionalidades en línea, aumentando su popularidad y, colapsando así, esa pila tradicional.

Como se puede ver en la Figura 2-6, las descargas suelen alcanzar el pico máximo poco después de su lanzamiento, y luego, cae rápidamente a una larga cola de jugadores dedicados. Para los juegos más populares, esto puede significar millones de personas en solo unos días o incluso horas. El aprovisionamiento para este tipo de tráfico resulta una tarea complicada.

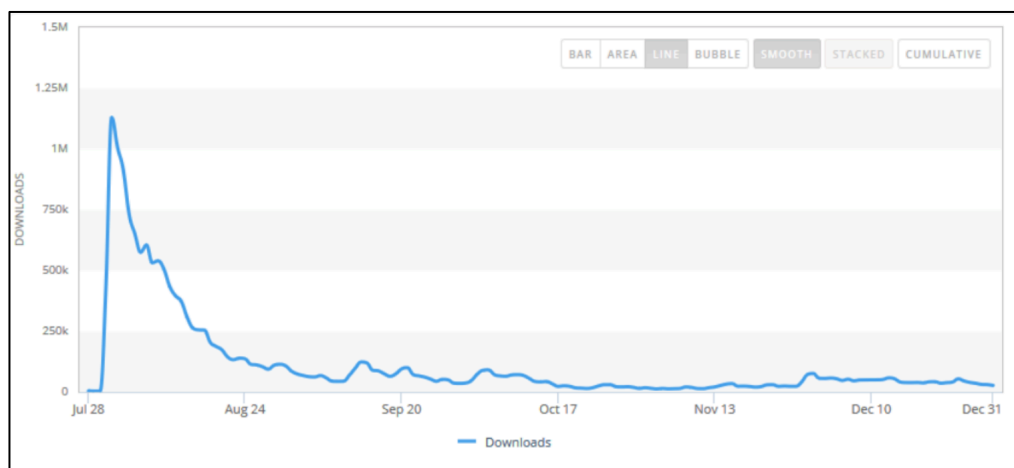


Figura 2-6 Curva de descargas del popular juego de móvil Crazy Taxi: City Rush [10]

De hecho, Yordan Gyurchev, director técnico de SEGA HARDlight declaró: “*Con estos volúmenes de datos masivos, incluso los cambios menores en la base de datos tienen un gran impacto. Para proporcionar una experiencia de juego perfecta, los desarrolladores deben estar íntimamente familiarizados con las compensaciones de rendimiento de la base de datos que están utilizando*”.

Esta industria sabía que los juegos iban a obtener todavía más funcionalidades en línea y que iban a generar ráfagas aun más masivas de actividad de los usuarios. Una buena parte de los datos de los juegos no encajaban en las filas y columnas de bases de datos relacionales. Para abordar estas limitaciones el equipo buscó alternativas, y la solución fue MongoDB.

Se llevaron a cabo una serie de evaluaciones y pruebas exhaustivas en múltiples dimensiones, tales como, coste, mantenimiento, monitoreo y copias de seguridad. MongoDB les brindó la capacidad de manejar y ejecutar juegos, incluso, cuando millones de personas estaban en línea. Ya no tenían por qué preocuparse del mantenimiento, las actualizaciones o las copias de seguridad y restauración de un punto determinado, lo que significaba que podían volver a un mismo punto de control con solo un clic.

El primer lanzamiento que brindó a los ingenieros sus primeras experiencias con MongoDB fue el Sonic Forces: Speed Battle, y fue todo un éxito. Era una aplicación exigente y activa que permitía una conexión constante a Internet y tablas de clasificación compartidas. Cabe reseñar que la base de datos se mantuvo completamente estable con bajas latencias y ni una sola interrupción del servicio [10].

Yordan concluye diciendo: “*Con MySQL, se habrían necesitado varios lanzamientos de juegos para que la base de datos fuera correcta. Con MongoDB, los grandes lanzamientos fueron un éxito desde el principio. Eso no es poca cosa*” [10].

2.1.2 InfluxDB

2.1.2.1 ¿Qué es InfluxDB?

InfluxDB es un sistema de gestión de bases de datos desarrollado por la empresa InfluxData. La última versión es InfluxDB 2.0, ya disponible como servicio en la nube, con una interfaz de usuario adaptable, basada en la web para registrar y visualizar los datos. InfluxData es un software de código abierto, el cual se puede usar de manera gratuita. La versión de pago InfluxData Enterprise, puede ser administrada por el usuario en una infraestructura local, nube privada, nube pública o perimetral y ofrece más funcionalidades, contratos de mantenimiento y controles de acceso especiales.

El sistema de gestión de bases de datos InfluxDB, cuyo logo puede verse en la Figura 2-7, fue desarrollado en Go, el lenguaje de programación de Google. Cuando se desarrolló la primera versión se usó el lenguaje de consulta InfluxQL para realizar consultas en bases de datos. Sin embargo, con la llegada de la nueva versión InfluxDB 2.0 se presentó el nuevo lenguaje de programación Flux.



Figura 2-7 Logotipo InfluxDB [122]

En la Figura 2-8, se puede ver un ejemplo de uso del lenguaje de programación Flux, donde se hace una consulta de datos almacenados desde la última hora, el filtrado por la medición de la *cpu* y la etiqueta *cpu=cpu-total*, la visualización de los datos en intervalos de 1 minuto, y el cálculo del promedio de cada ventana.

```
from(bucket:"example-bucket")
  |> range(start:-1h)
  |> filter(fn:(r) =>
    r._measurement == "cpu" and
    r.cpu == "cpu-total"
  )
  |> aggregateWindow(every: 1m, fn: mean)
```

Figura 2-8 Ejemplo del lenguaje de programación Flux [103]

Flux es un lenguaje independiente de *scripts* y consultas para bases de datos de series temporales, TSDB (*Time Series Database*). Se puede utilizar a partir de la versión InfluxDB 1.7 o de forma independiente. También es posible usarlo en combinación con otras bases de datos.

Este lenguaje ha sido optimizado para el proceso ETL (*Extract, Transform and Load*), es decir, que tiene la capacidad de extraer, transformar y cargar la información en bases de datos, aunque no es compatible con el lenguaje de consulta InfluxQL. Sin embargo, ya que hay muchos clientes que usaban anteriormente el lenguaje InfluxQL, el proveedor tiene previsto desarrollar una ruta de migración permitiendo así la traducción del antiguo lenguaje a Flux, facilitándose su uso.

Flux se basa en el popular lenguaje de *scripts* de JavaScript por lo que es fácil de aprender, ofreciendo una gran flexibilidad en su uso. Una de las características más llamativas de Flux es que permite compatibilidad con diferentes fuentes de datos. De tal manera que puede trabajar con otras herramientas de análisis. [11].

InfluxData ofrece una solución en la nube con InfluxDB Cloud 2.0 para Amazon Web Services (AWS), Google Cloud Platform (GCP) o Microsoft Azure. En el caso de la denominada computación sin servidor (Serverless-Computing) el cliente no necesita una infraestructura de servidor propia.

En la nube no es necesario reservar servidores individuales, ya que el sistema es capaz de adaptarse automáticamente según la carga de actividad que hay en cada momento. Esta característica supone una gran ventaja en aplicaciones industriales del IoT o aprendizaje automático, debido a la tendencia que hay a la oscilación de los volúmenes de datos.

En cuanto a la variante local, la cual se instala en un servidor propio, se concentra todo el sistema de gestión de bases de datos en una sola aplicación, que actualmente, está únicamente disponible en Linux de 64 bits, Linux para procesadores ARM, MacOS y Docker como contenedor [11].

Como puede verse en la Figura 2-9, actualmente InfluxDB es una de las herramientas más utilizadas en el ámbito de la gestión de bases de datos de series temporales. Según la página web DB-Engines [12], InfluxDB es la herramienta más popular en este sector a día de hoy, además, ha ostentado este título durante varios años consecutivos, lo que constata que es una herramienta muy competente en el mundo de los flujos de datos.

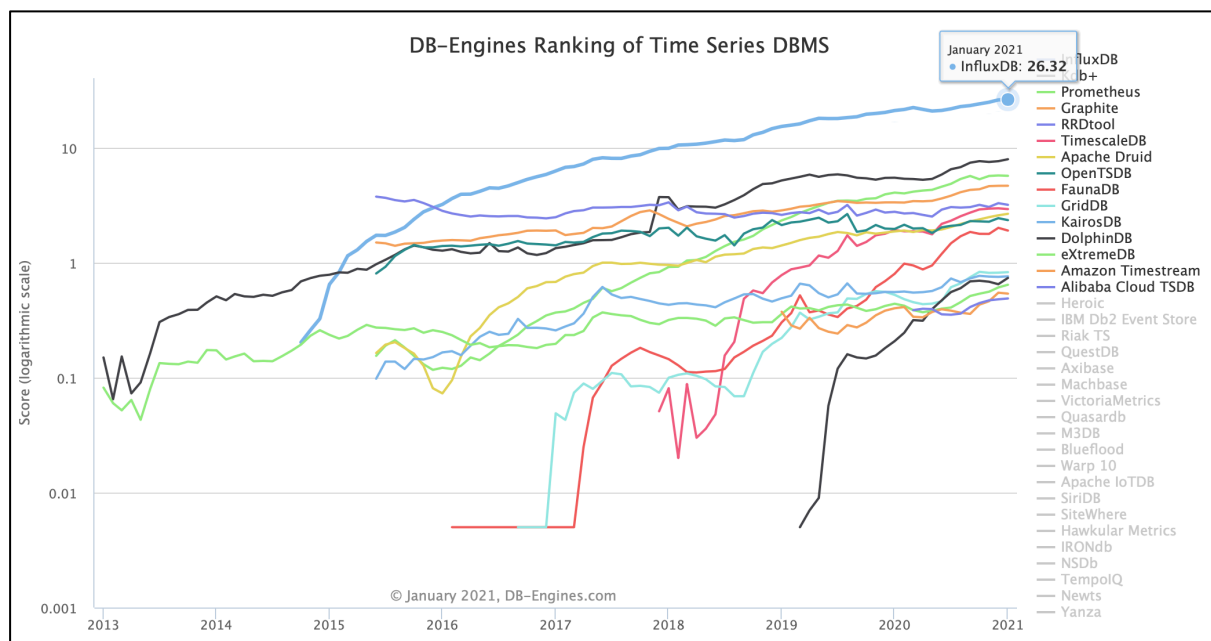


Figura 2-9 Gráfica de los DBMS de datos temporales según su popularidad [12]

2.1.2.2 Funcionamiento de InfluxDB

InfluxDB ha sido concebido para bases de datos que almacenan series temporales. Usándose para analizar datos de sensores o protocolos con marcas temporales durante un periodo de tiempo determinado. En estos casos, entran millones de datos en juego, como los que proporcionan los equipos del IoT (*Internet of Things*) o los instrumentos científicos de medición por medio de un flujo continuo de datos, como el sistema AIS.

Es por ello, que InfluxDB cuenta con un servicio de tiempo que usa el *Network Time Protocol* (NTP) para garantizar así, que todos los sistemas estén sincronizados al mismo tiempo.

En la Figura 2-10 se observa una aplicación de cómo serían las columnas de datos (cabe reseñar que permite muchas más columnas), observamos, la fuente de donde se extraen los datos, el valor de los datos en sí y la marca temporal correspondiente.

Sensor	Valor	Hora
Sensor 1	140,50	23/04/2020 @ 10:00
Sensor 2	110,02	23/04/2020 @ 10:00
Sensor 1	142,32	23/04/2020 @ 10:05
Sensor 2	110,50	23/04/2020 @ 10:05
...	...	...

Figura 2-10 Ejemplo flujo de datos en InfluxDB [11]

En InfluxDB se puede distinguir entre *tags* y *fields*. Los *tags* contienen metadatos incluidos en el índice, mientras que los *fields* incluyen valores que pueden ser evaluados posteriormente. En la Figura 2-10, la primera columna se corresponde con un *tag* y, la segunda, con un *field*. Facilitando esta distinción, el manejo de datos y la evaluación de la medición que se obtenga.

La gran ventaja que ofrece, es la rapidez a la hora de almacenar y procesar datos de medición con marcas temporales. Ya que, a diferencia de otros sistemas de gestión de bases de datos (DBMS, *DataBase Management System*), que tienen que dedicar parte de su rendimiento a la organización de un índice complejo, InfluxDB puede mantener una rápida velocidad de escritura gracias a que utiliza un índice muy sencillo.

Esta plataforma, logra el tratamiento de grandes cantidades de datos temporales, desde el momento de su recolección, hasta su análisis final. Siendo una alternativa muy útil para trabajar en escenarios de BigData de series temporales, consiguiendo una gran optimización en su almacenamiento y acceso.

Un *stack* es una lista de todos los servicios tecnológicos utilizados para construir y ejecutar una aplicación. Sabiendo este concepto, el *stack* de InfluxData está dividido en 4 elementos principales: Telegraf, InfluxDB, Chronograf y Kapacitor, como se puede ver en la Figura 2-11. Cabe reseñar que todas las herramientas son de código libre [13].

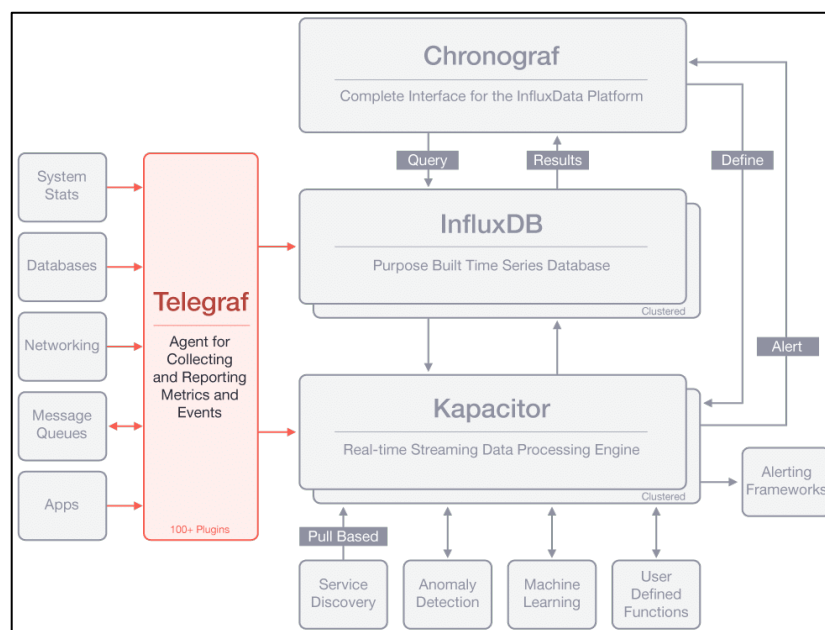


Figura 2-11 Stack InfluxDB [13]

Se va a proceder a explicar la función de cada uno de los elementos principales del *stack* [13]:

- **Telegraf:** Es un agente que se encarga de recopilar y reportar métricas y/o eventos desde cualquiera de las fuentes posibles. Cuenta con más de 100 *plugins* para diferentes orígenes de datos. Además, permite añadir fácilmente nuevos orígenes de datos ya sean con los *plugins* existentes para orígenes de datos muy conocidos, o hechos a medida.
- **InfluxDB:** Es la base de datos encargada de la persistencia, indexación y búsqueda de datos. Además, está especializada en tratamiento de datos de series temporales. Puede manejar gran cantidad de inserciones y búsquedas, y está pensada para tratar datos de métricas, monitoreo, sensores del IoT y análisis en tiempo real. Dos características reseñables son, que permite la configuración de políticas de retención, RP (*Retention Policies*), es decir, que tiene la capacidad de expirar automáticamente datos antiguos o modificar el número de réplicas; y también puede lanzar colas continuas, CQ (*Continuous Queues*) para reducir la resolución de los datos. Un ejemplo de este último sería, cambiar el cálculo de la media de un valor que tiene una resolución de 10 minutos a una resolución de 30 minutos para datos de hace más de un mes.
- **Chronograf:** Se encarga de la visualización y monitorización de datos. Permite ver y controlar el estado de todos los servidores del *cluster*. Además, ofrece una interfaz gráfica para configurar alertas, ejecutar *jobs* y detectar posibles anomalías en los datos, utilizando las funcionalidades de Kapacitor. También permite crear *dashboards* personalizados para la visualización de los datos con distintos tipos de gráficos y, administrar la base de datos y las políticas de retención, con la posibilidad de ver las colas en curso y cancelarlas, y administrar usuarios. La herramienta Grafana para visualización de datos es muy utilizada con este fin.
- **Kapacitor:** Es el elemento que facilita la configuración de alertas, ejecución de *jobs* y detección de anomalías en los datos. Permite tratar datos en *streaming*, ejecutar transformaciones de datos y guardarlos en InfluxDB y, configurar funciones personalizadas para la detección de posibles anomalías en los datos.

2.1.2.3 Escenarios de uso

InfluxDB es una herramienta de gestión de datos muy utilizada a nivel mundial por compañías de alto nivel. Analizando el uso de la herramienta, se han obtenido los siguientes valores:

- Resultados en Google al buscar “InfluxDB”: 4.230.000
- Seguidores en la red social Twitter (@InfluxDB): 20.782 seguidores
- Fecha de búsqueda: 19 de enero de 2021

InfluxDB ofrece una gran versatilidad y es usado en múltiples escenarios de la industria como el de las finanzas, la energía, las telecomunicaciones y la tecnología. Algunas de las empresas más punteras hoy en día, utilizan los servicios de esta herramienta [14].

En el mundo de las finanzas destaca PayPal, una empresa estadounidense que desarrolló una plataforma de pago digital que admite transferencias de dinero en línea, empleándose como alternativa electrónica al método tradicional en papel para enviar y recibir dinero. PayPal tiene actualmente 277 millones de titulares de cuentas activas en todo el mundo; está disponible en más de 200 mercados y utiliza alrededor de 100 monedas diferentes, lo que supone una gran cantidad de datos.

Esta empresa necesitaba encontrar una solución de monitoreo de *hosts* escalable de extremo a extremo. Según iba creciendo la empresa, fueron modernizando su infraestructura y realizando una transición de muchas de sus aplicaciones a una arquitectura basada en contenedores. Necesitaban una nueva solución de monitoreo para trabajar con contenedores y proporcionar recopilación, almacenamiento, alertas y visualización de métricas, todo a la vez.

Para ello, la solución de monitoreo tenía que ser escalable para seguir el ritmo de la infraestructura de la empresa. PayPal tiene nueve centros de datos, con 30.000 instancias cada uno, y todos ellos con

sus propios *clusters*. La empresa estaba migrando todas sus aplicaciones de 20 años compiladas en C++ a contenedores y sistemas operativos mucho más modernos, con gran parte de sus hipervisores Docker alojando de 50 a 100 contenedores al mismo tiempo [14].

Para esta nueva solución, PayPal estableció cuatro requisitos técnicos que cumplir a través de un proveedor [15]:

- Un agente confiable y extensible, que se encuentre en todos los sistemas para monitorear las métricas básicas del sistema operativo, como CPU, disco, memoria, aplicaciones de terceros y bases de datos.
- *Backend* de base de datos de series temporales para el historial de informes.
- Capacidad para monitorear múltiples contenedores *Docker* con un solo agente, siendo fundamental para mantener la sobrecarga del agente en todo el sistema.
- Alerta inteligente basada en datos de series temporales.

PayPal eligió InfluxDB Enterprise de InfluxData, aprovechando todos los componentes de la plataforma central de código abierto de InfluxData para crear una solución utilizando agregadores, colas de mensajes y editores de Telegraf (véase Figura 2-12) con el fin de controlar el tamaño de la carga útil de datos, administrar el flujo de mensajes y evitar el punto único de fallo.

InfluxDB Enterprise desarrolló una solución de extremo a extremo como la que PayPal estaba buscando. Utilizando la plataforma InfluxData, PayPal pudo crear una solución de monitoreo resistente que funcionaba a escala. Durante el proceso, el equipo de Monitoreo de Ingesta/Recolectores y Plataformas de alerta de PayPal, desarrolló las mejores prácticas formuladas para escalar InfluxDB Enterprise en su entorno. De hecho, el director de este equipo presentó una charla en 2018 sobre las mejores prácticas para escalar un *cluster* empresarial de InfluxDB [15].

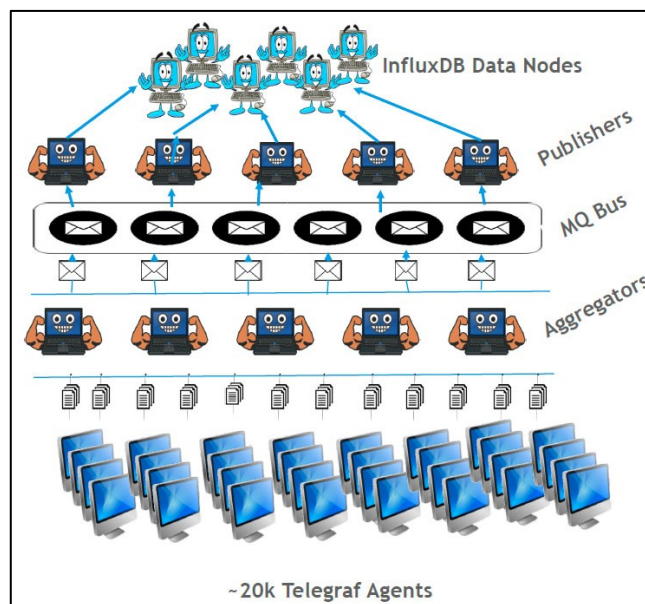


Figura 2-12 Arquitectura técnica usando InfluxDB Enterprise y Telegraf Agents [15]

2.1.3 KsqlDB

2.1.3.1 ¿Qué es KsqlDB?

KsqlDB es una base de datos de eventos en *streaming*, desarrollada por la compañía Confluent y distribuida con licencia de código libre. En la Figura 2-13 puede verse su logotipo [16].

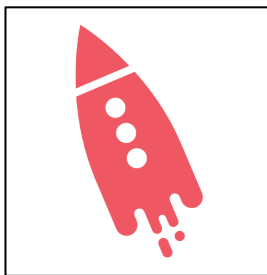


Figura 2-13 Logotipo KsqlDB [121]

La función principal de una base de datos de eventos en *streaming*, es ayudar a crear aplicaciones de procesamiento de flujo. Lo que hace es consolidar la gran cantidad de componentes que hay en prácticamente todas las arquitecturas de eventos en *streaming*.

La mayoría de arquitecturas de *streaming* actuales son soluciones fragmentadas improvisadas a partir de diversos proyectos, estas aplicaciones necesitan como mínimo un subsistema para adquirir eventos de fuentes de datos existentes, otro para almacenarlos, otro para procesarlos y un último para atender consultas contra materializaciones agregadas. Cada aplicación tiene su propia arquitectura y resultan muy complejas algunas de ellas [17].

KsqlDB proporciona un modelo mucho más sencillo, sin embargo, depende de Apache Kafka, herramienta de la que se hablará en el Apartado 2.1.3.2.

Permite desarrollar aplicaciones de *streaming* sobre un sistema escalable, flexible y tolerante a fallos, proporcionando una interfaz SQL (*Structured Query Language*) para procesamiento de flujos de datos con operaciones de filtrado, transformación, agregación y posibilidad de usar ventanas de procesamiento. En la Figura 2-14, se puede observar un ejemplo del lenguaje de consulta de KsqlDB, donde se crea una tabla. Además, esta herramienta proporciona muy bajas latencias y permite guardar el estado del sistema [16].

```

1  -- table with declared columns:
2  CREATE TABLE users (
3      id BIGINT PRIMARY KEY,
4      usertimestamp BIGINT,
5      gender VARCHAR,
6      region_id VARCHAR
7  ) WITH (
8      KAFKA_TOPIC = 'my-users-topic',
9      VALUE_FORMAT = 'JSON'
10 );

```

Figura 2-14 Ejemplo del lenguaje de consulta de KsqlDB [77]

2.1.3.2 Qué es Apache Kafka

Apache Kafka nació como alternativa a las colas de mensajería tradicionales, tales como Artemis o RabbitMQ. Fue desarrollada por la compañía LinkedIn para dar solución al principal problema que tenían, procesar billones de mensajes al día. Tras finalizar su desarrollo, en 2014 LinkedIn donó esta herramienta a Apache Foundation convirtiéndose en un proyecto de código abierto.

Gracias a su arquitectura, Apache Kafka es capaz de manejar billones de mensajes al día de forma totalmente eficiente, dicha arquitectura está formada por tres elementos fundamentales y basada en una plataforma de datos de tipo publicador/consumidor (véase Figura 2-15):

- **Publicador:** Es aquel que genera el dato o fuente de datos. Por ejemplo, Twitter.
- **Consumidor:** El que lee el dato, pueden ser uno o varios.
- **Topic:** Donde quedan guardados los datos producidos por el publicador para que el consumidor los lea.

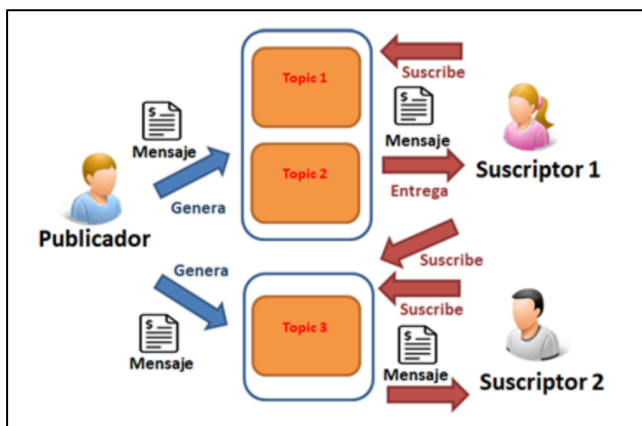


Figura 2-15 Arquitectura básica Apache Kafka [18]

Como se puede ver en la Figura 2-14, tenemos un publicador, el cual genera la información que se escribe en el *Topic*, de manera que el consumidor puede acceder a ella a demanda. Puede haber millones de *Topics* almacenados de manera particionada para aumentar el rendimiento de procesamiento en su lectura y escritura.

Una de las ventajas de Kafka sobre las colas de mensajería tradicionales, es que la misma información puede ser consumida por diferentes consumidores o suscriptores. Además, cada consumidor que necesite información no interfiere en los demás, pudiendo ser utilizada por las diferentes aplicaciones tantas veces como se necesite [18].

2.1.3.3 Funcionamiento de KsqlDB

KsqlDB transforma continuamente flujos de datos, obteniendo *topics* de Apache Kafka existentes, filtrándolos, procesándolos y reaccionando a ellos para crear nuevos *topics* derivados. Estos *topics* pueden ser eventos puros, o actualizaciones de tablas con clave que se estén formando fuera del flujo, como podría ser, unir muchas fuentes de datos sobre ciertos clientes para así crear un perfil de cliente unificado que esté continuamente actualizado. KsqlDB es capaz de procesar continuamente el flujo de eventos entrantes e ir actualizando las tablas.

Hasta ahora, KsqlDB solo tenía soporte para este tipo de consulta continua y en *streaming* contra sus tablas de datos, como en el ejemplo anterior de los clientes, donde la salida es un flujo continuo de perfiles de clientes, pero no se puede realizar una búsqueda en un perfil en particular. Sin embargo, con la nueva versión, es posible hacer búsquedas tradicionales sobre estas bases de datos, con el nombre de consultas de extracción ya que permiten extraer datos en un momento determinado.

La siguiente arquitectura (Figura 2-16) representa un procesamiento de flujo común de una aplicación:

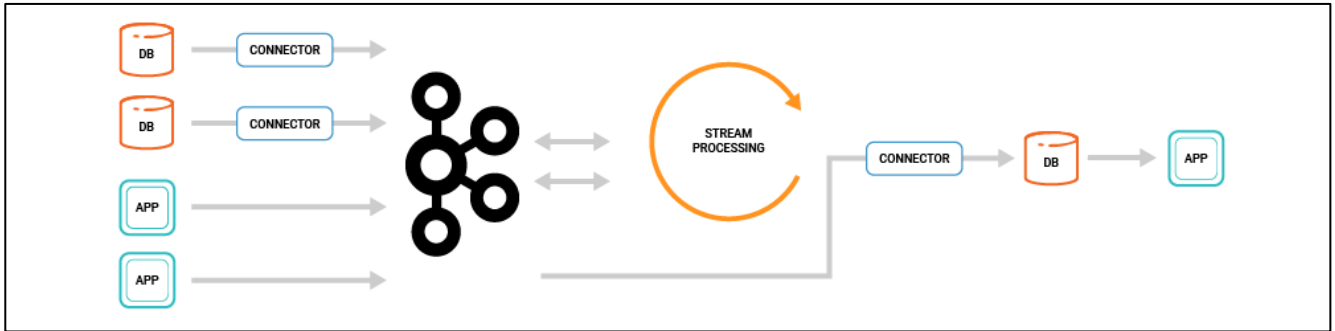


Figura 2-16 Arquitectura procesamiento de flujo común [19]

La arquitectura de la Figura 2-17, está formada por varias partes móviles que se describen brevemente a continuación [19].

- Extraer: Hay una serie de conectores que pueden estar ejecutándose en Kafka Connect, que se encargan de extraer datos de bases de datos de origen u otros sistemas.
- Almacenar flujos: En Kafka se almacenan flujos de eventos y flujos derivados.
- Transformar: Los flujos son transformados en sistemas de procesamiento de flujo tales como Kafka Streams.
- Carga: Se produce una carga continua en la cola, posiblemente Kafka.
- Almacenar y consultar: Existe una base de datos tradicional que recibe el flujo, se calculan los resultados y los entrega a la aplicación correspondiente.

En la Figura 2-17 se pueden ver mejor las diferentes etapas de la arquitectura del procesamiento de un flujo común dividido en bloques.

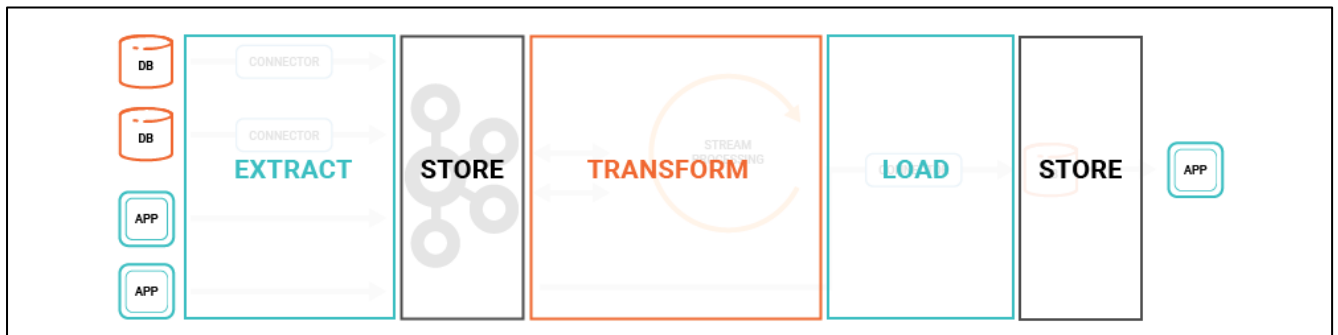


Figura 2-17 Arquitectura por bloques [19]

Como puede observarse, es bastante complejo, ya que hay involucrados varios sistemas, los cuales deben integrarse, asegurarse, monitorearse y operarse como uno solo. Es una arquitectura muy pesada para una aplicación, es por ello, que algunos optan por arquitecturas simples de tres niveles de las aplicaciones CRUD tradicionales, las cuales solo necesitan una única base de datos. Sin embargo, KsqlDB proporciona la misma simplicidad para las aplicaciones de transmisión de eventos que las bases de datos relacionales CRUD.

Con KsqlDB se simplifica sustancialmente esa arquitectura, ya que solo hay dos elementos en juego, Kafka y KsqlDB (véase Figura 2-18), los cuales reúnen el conjunto completo de componentes necesarios, hasta el procesamiento y las consultas [19].

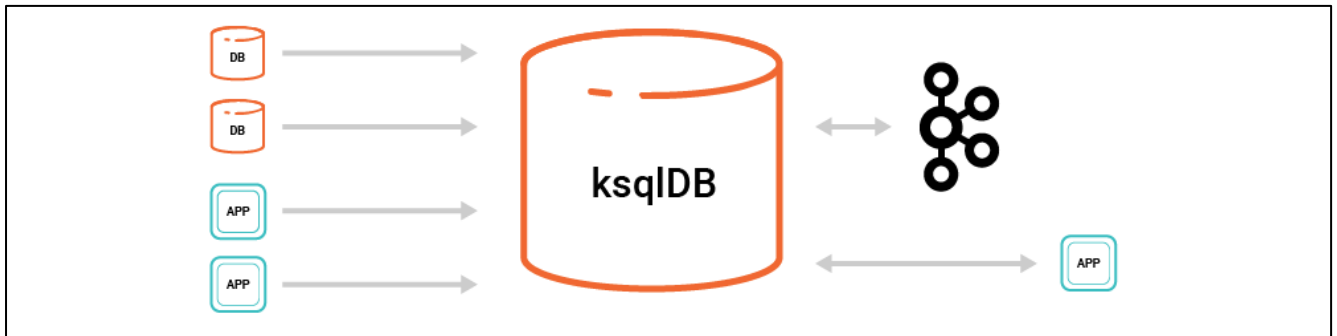


Figura 2-18 Arquitectura con Kafka y KsqlDB [19]

Con KsqlDB es posible crear una aplicación de eventos en *streaming* extremo a extremo con solo unas pocas consultas SQL [19]:

- Extraer los flujos de eventos de los sistemas de origen a través de Kafka Connectors.
- Transformar mediante la materialización continua de los resultados de los flujos.
- Consultas como acceso a resultados particulares y ofrecer una interfaz de usuario.

A partir de esto, es posible plantear la siguiente pregunta, ¿Es posible integrar las capacidades de procesamiento de flujo y de consulta? En principio, una base de datos distribuida y una capa de procesamiento de flujo pueden parecer muy diferentes, sin embargo, una vez que se entiende el funcionamiento de ambas, no lo son tanto.

La arquitectura de las bases de datos distribuidas se centra en un registro de confirmación distribuido para sincronizar los datos entre los nodos (véase Figura 2-19). Está formada por un conjunto de nodos de almacenamiento, los cuales mantienen los datos para que sean consultables en cualquier acceso local aleatorio. Además, las réplicas ofrecen tolerancia a fallos, permitiendo que, si un nodo falla, sus datos se mantengan en un nodo idéntico en otro lugar [19].

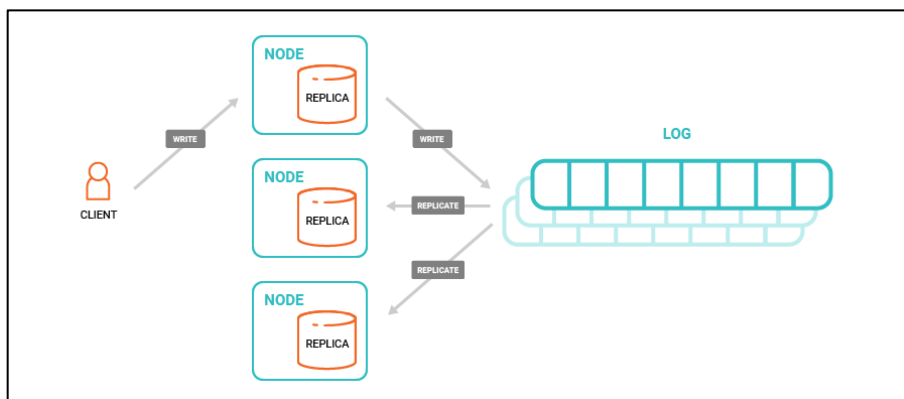


Figura 2-19 Arquitectura interna [19]

Este es precisamente el modelo para administrar estados en KsqlDB y Kafka Streams, utilizando RocksDB [20], que ofrece un formato local de consulta en disco a través de un registro de confirmación, el cual secuencia las actualizaciones de estado proporcionando tolerancia a fallos entre las instancias, para conseguir una alta disponibilidad.

Cabe destacar que KsqlDB es capaz de realizar consultas *push*, para que las aplicaciones procesen y reaccionen continuamente a un sinfín de eventos. Esto se debe a que, en una empresa moderna, las aplicaciones están conectadas entre sí, de tal forma que lo que sucede en un extremo del negocio puede

desencadenar una actividad en otra parte de este. De hecho, en las empresas modernas, el usuario puede llegar a ser una aplicación de software en vez de un humano, de manera que mientras que un humano usa la aplicación momentáneamente, un servicio es capaz de ejecutarse de forma continua, reaccionando, procesando y transformando los eventos a medida que van llegando. [19]

2.1.3.4 Escenario de uso

Esta herramienta no muy popular a día de hoy, de hecho, no aparece en el ranking de DB-Engines [12]. Sin embargo, hay grandes empresas que sí que han optado por esta tecnología, como la empresa alemana Bosch, cuyo caso se explicará a lo largo de este apartado. Se han obtenido los siguientes resultados tras un análisis de su uso:

- Resultados en Google al buscar “KsqlDB”: 82.700
- Seguidores en la red social Twitter (@ksqlDB): 2.868 seguidores
- Fecha de búsqueda: 23 de enero de 2021

La empresa alemana es una de las muchas que a día de hoy utilizan KsqlDB para impulsar una transformación digital a gran escala, implementando una arquitectura común de IoT que admita el *streaming* en tiempo real, basada en eventos y el almacenamiento persistente de eventos.

El Grupo Bosch es un proveedor a nivel mundial de tecnología y servicios, que trabaja principalmente en cuatro sectores comerciales: Movilidad, tecnología industrial, bienes de consumo y energía y tecnología de la construcción. Actualmente, Bosch ha invertido mucho en IoT, ofreciendo soluciones innovadoras para hogares inteligentes, movilidad conectada y fabricación conectada. Utiliza toda su experiencia en tecnología de sensores, software y servicios, así como su propia nube de IoT.

Bosch Power Tools es una división de The Bosch Group que se encarga de proveer herramientas eléctricas, accesorios y herramientas de medición. La empresa se vio envuelta en una transformación digital, teniendo como objetivo, personalizar la experiencia del usuario en cuanto a sus productos con una solución de IoT que aprovechaba los datos de los sensores de herramientas eléctricas inteligentes y una aplicación móvil [21].

Por ello, Bosch Power Tools creó una plataforma común mediante Confluent Cloud Enterprise con Apache Kafka y KsqlDB, permitiendo la comunicación asíncrona de las aplicaciones y micro-servicios con transmisión en tiempo real basada en eventos y almacenamiento persistente de eventos.

Esta empresa ya ha implementado varias aplicaciones nuevas basadas en esta arquitectura y tienen más proyectos planificados, e incluso, en proceso. Al incorporar capacidades del motor SQL de transmisión de KsqlDB de Confluent y Kafka Stream, estas aplicaciones admitirán alertas en tiempo real y nuevos paneles de control en tiempo real, los cuales presentarán y fusionarán datos procedentes de plantas de fabricación, distribuidores, propietarios de herramientas y otras fuentes de la empresa. Muy pronto, Bosch podrá recibir multitud de datos de las herramientas después de su venta.

Los resultados de optar por Confluent con Kafka y KsqlDB, se han visto reflejados en una plataforma de transmisión de eventos escalable y de bajo mantenimiento.

2.1.4 Elasticsearch

2.1.4.1 ¿Qué es Elasticsearch?

Es un motor de búsqueda que permite realizar consultas y análisis casi en tiempo real para todo tipo de datos. Puede trabajar con datos que tengan texto estructurado o no estructurado, datos numéricos o datos geoespaciales. Además, Elasticsearch puede almacenarlos e indexarlos de manera eficiente de forma que admita búsquedas rápidas. Esta herramienta puede ir mucho más allá de la simple recuperación de datos, ya que es capaz de agregar información para descubrir tendencias y patrones en sus datos. A medida que crece el volumen de datos y consultas, la naturaleza distribuida de Elasticsearch

permite que su implementación crezca sin problemas, ofreciendo así una gran escalabilidad. Puede verse su logotipo en la Figura 2-20 [22].



Figura 2-20 Logotipo Elasticsearch [138]

Elasticsearch ofrece una gran velocidad y flexibilidad, además, es capaz de manejar datos en una gran variedad de casos de uso, los cuales se exponen a continuación [22]:

- Agregar un cuadro de búsqueda a una aplicación o sitio web.
- Almacenar y analizar registros, métricas y datos de eventos de seguridad.
- Modelar automáticamente el comportamiento de los datos en tiempo real mediante el aprendizaje automático.
- Automatizar flujos de trabajo comerciales.
- Administrar, integrar y analizar información espacial utilizando Elasticsearch como un sistema de información geográfica.
- Almacenar y procesar datos genéticos utilizando Elasticsearch como una herramienta de investigación bioinformática.

Las principales características de Elasticsearch son las siguientes [23]:

- Su escalabilidad horizontal, ya que el diseño de su arquitectura se basó en ello.
- Tiene latencias muy bajas aproximándose al tiempo real, del orden de milisegundos desde que se indexa hasta que el dato está disponible para la búsqueda.
- Ofrece una alta disponibilidad al tener replicación de datos en nodos diferentes, por lo que es tolerante a fallos en estos.
- Dispone de funciones de búsqueda de texto completo. En bases de datos convencionales solo se consideran campos concretos para la búsqueda como el título o las referencias, sin embargo, en Elasticsearch se considera todo el contenido de los documentos.
- Está orientado a documentos JSON.
- No requiere esquemas.
- Tiene un desarrollo amigable con unas API (*Application Programming Interfaces*) sencillas.

Además, como se puede ver en la Figura 2-21, según la clasificación de bases de datos realizada por la página web de DB-Engines [24], Elasticsearch ocupa el primer puesto entre las bases de datos de búsqueda más utilizadas, por lo que es una herramienta muy competente y popular en este ámbito [25].

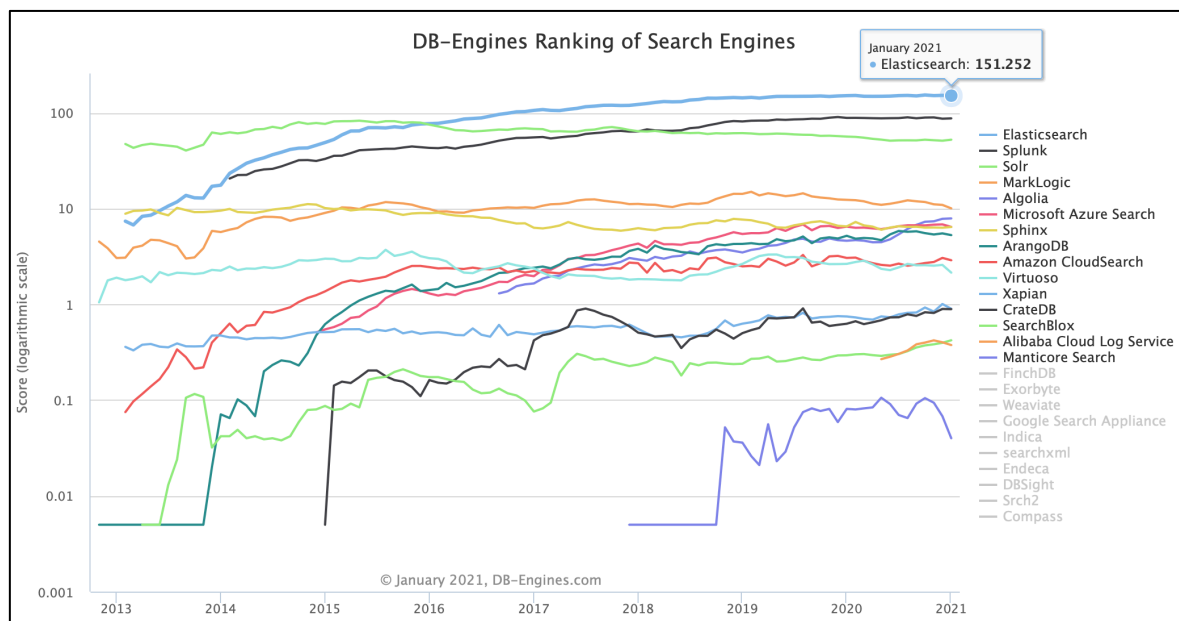


Figura 2-21 Gráfica de bases de datos de búsqueda según su popularidad [25]

2.1.4.2 Funcionamiento de Elasticsearch

Elasticsearch es un almacén de documentos distribuido. En lugar de almacenar la información en tablas de datos, esta herramienta almacena estructuras de datos complejas que se han serializado como documentos JSON. Cuando en un *cluster* hay varios nodos de Elasticsearch, los documentos almacenados se van distribuyendo por el *cluster*, de tal forma que se puede acceder a ellos inmediatamente desde cualquier nodo.

Al almacenar un documento, se indexa y se puede buscar por completo prácticamente en tiempo real. Elasticsearch utiliza una estructura de datos llamada índice invertido, permitiendo una búsqueda rápida de textos completos. Este índice, enumera cada palabra única en cualquier documento e identifica todos los documentos en los que aparecen esas palabras.

Un índice es una colección optimizada de documentos, donde cada documento es una colección de campos, siendo estos los pares clave-valor que contienen sus datos. Elasticsearch indexa todos los datos en cada campo, y cada campo indexado tiene una estructura de datos dedicada y optimizada de forma predeterminada. La capacidad de usar estructuras de datos por campo para ensamblar y devolver resultados de búsqueda, es lo que hace tan rápida a esta herramienta.

Elasticsearch no tiene esquema, lo que quiere decir que los documentos se pueden indexar sin especificar explícitamente cómo manejar cada uno de los diferentes campos que pueden ocurrir en un documento. Esta herramienta tiene la capacidad de habilitar un mapeo dinámico para detectar y agregar automáticamente nuevos campos al índice, logrando facilitar la exploración e indexación de sus datos; al comenzar a indexar documentos, la herramienta será capaz de detectar y asignar valores booleanos, fechas y cadenas a los diferentes tipos de datos.

Es posible controlar la asignación dinámica y definir explícitamente las asignaciones para tomar el control de cómo se almacenan e indexan los campos. Al optar por una solución más personalizada donde sea posible definir las asignaciones deseadas, es posible [26]:

- Distinguir entre campos de cadena de texto completo y campos de cadena de valor exacto.
- Realizar análisis de textos en un lenguaje específico.
- Optimizar campos para coincidencias parciales.
- Utilizar formatos de fecha personalizados.

- Utilizar tipos de datos como *geo_point* y *geo_shape*, los cuales no se pueden detectar automáticamente.

A la hora de indexar el mismo campo, Elasticsearch ofrece una gran flexibilidad, ya que, es posible indexar un campo de cadena, como un campo de texto para la búsqueda de texto completo y como un campo de palabra clave para ordenar o agregar sus datos, o también es posible utilizar más de un analizador de lenguajes para procesar el contenido de un campo de cadena.

La cadena de análisis que se aplica a un campo de texto completo durante la indexación es similar a la que se utiliza en el momento de la búsqueda. Al consultar un campo de texto completo, el texto de la consulta se ve sometido al mismo análisis antes de que los términos se busquen en el índice.

Por otro lado, Elasticsearch proporciona una API REST (*Representational State Transfer*) para administración del *cluster*, indexado y búsqueda de datos de la manera más simple y coherente posible. La API REST admite consultas estructuradas, consultas de texto completo, y la combinación de ambas. Las consultas estructuradas son muy similares a las que se pueden hacer en SQL, por ejemplo, es posible buscar los campos de género y edad en un índice de empleados y ordenar las coincidencias por el campo de fechas de contratación. Por otro lado, las consultas de texto completo son capaces de encontrar todos los documentos que coinciden con la cadena de consulta y los devuelven ordenados por relevancia, según la similitud que tengan los términos de búsqueda. En la Figura 2-22, se puede observar un ejemplo del lenguaje de consulta de Elasticsearch, donde se realiza una búsqueda en un índice de todos aquellos empleados que tengan 40 años y sean del estado “ID”

```
GET /bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "ID" } }
      ]
    }
  }
}
```

Figura 2-22 Ejemplo del lenguaje de consulta de Elasticsearch [135]

En el caso de tener datos geospaciales, Elasticsearch es capaz de indexar datos no textuales en estructuras de datos optimizadas, las cuales admiten consultas geográficas y numéricas de alto rendimiento.

Las agregaciones permiten crear resúmenes complejos de los datos, obteniendo de esta forma información sobre métricas, patrones y tendencias clave. Además, pueden operar junto con las solicitudes de búsqueda, siendo posible buscar documentos, filtrar resultados y realizar análisis al mismo tiempo, en los mismos datos y con una sola solicitud. Las agregaciones se calculan en el contexto de una búsqueda en particular, por ejemplo, al hacer una búsqueda de las agujas de bordar antiadherentes de tamaño 70, no se mostraría únicamente un recuento de todas las agujas que pueda haber de ese tipo, sino que se muestra un recuento de todas las agujas de tamaño 70 que coinciden con los criterios de búsqueda de los usuarios [27].

En cuanto a su arquitectura, está diseñado para estar siempre disponible y adaptarse a las necesidades que puedan surgir. Es capaz de agregar servidores, que serían los nodos, a un *cluster* para aumentar la capacidad, de tal forma que puedan distribuirse automáticamente los datos y la carga de las consultas en

todos los nodos disponibles. Elasticsearch sabe cómo equilibrar los *clusters* de múltiples nodos para garantizar la mayor escalabilidad y disponibilidad.

Los índices no son más que agrupaciones lógicas de uno o más fragmentos (*shards*) físicos, donde cada *shard* es en realidad un índice autónomo. Distribuyendo los documentos en un índice a través de varios *shards* y repartiendo estos en varios nodos, es posible garantizar una redundancia que proteja contra fallos de hardware, y aumentar la capacidad de consulta a medida que se van agregando nuevos nodos al *cluster*. Elasticsearch va migrando automáticamente los *shards* a medida que crece el *cluster* para reequilibrarlo.

Existen dos tipos de *shards*: primarios y réplicas. Cada documento de un índice se corresponde con un nodo principal, mientras que un *shard* réplica es una copia de un *shard* principal. Las réplicas son copias redundantes de los datos de tal forma que estos quedan protegidos contra fallos de hardware y, además, aumenta la capacidad para atender solicitudes de lectura como buscar y recuperar un documento.

Cabe reseñar que el número de *shards* primarios queda fijado desde el momento en que se crea el índice, mientras que el número de *shards* réplica, se puede modificar en cualquier momento sin que se vean afectadas las operaciones de indexado o consulta [28].

A continuación, en la Figura 2-23 se puede ver un ejemplo gráfico de cómo funciona el almacenamiento en Elasticsearch.

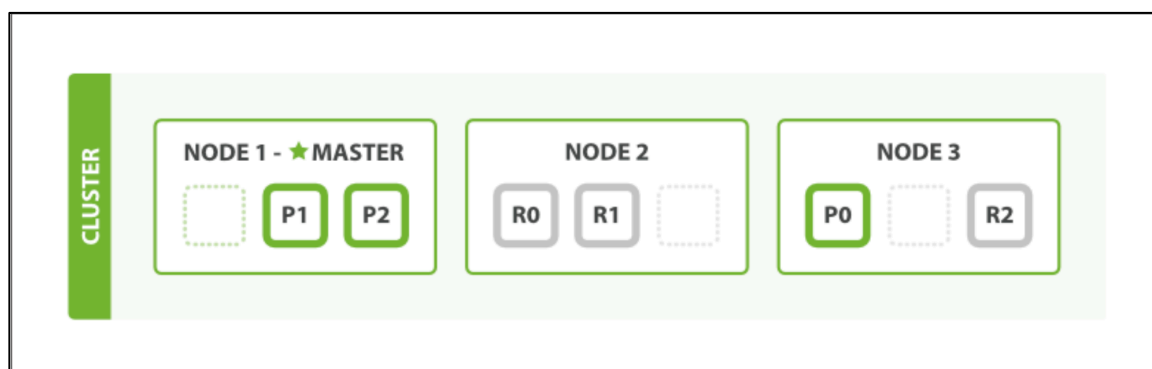


Figura 2-23 Índice dividido en dos *shards* (P0 y P1), 3 nodos y 2 réplicas por *shard*. [23]

En este caso se puede observar un índice dividido en dos *shards*, denominados P0 y P1, con tres nodos y dos réplicas por cada *shard*, donde los marcados en verde son los primarios y los grises las réplicas.

De tal forma que, si falla uno de los nodos, las réplicas pasarían a ser primarias, y al tener dos *shards* menos, se volverían a generar nuevas réplicas, como se puede ver en la Figura 2-24.

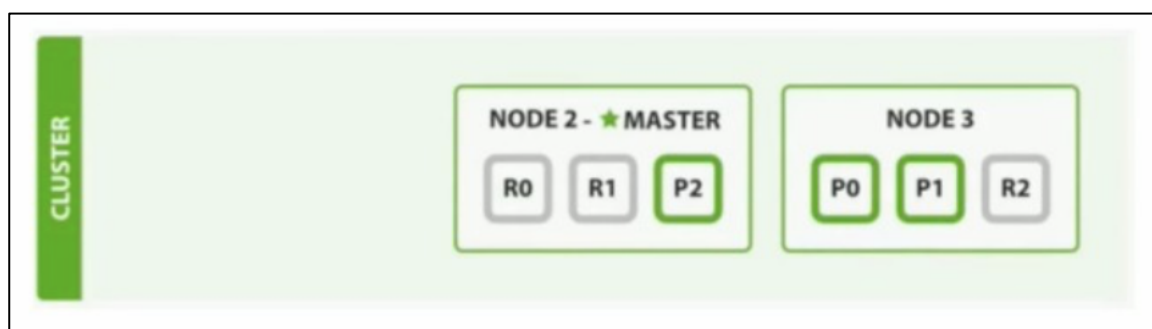


Figura 2-24 Fallo de un nodo [23]

Tras el fallo del nodo 1, el nodo 2 asume su puesto y el sistema seguirá funcionando, aunque con una menor eficiencia [23].

2.1.4.3 Escenario de uso

Elasticsearch está en primer lugar en la clasificación que hace DB-Engines [25] sobre bases de datos de búsqueda. Varias empresas importantes han optado por esta herramienta, como Lenovo, de la cual hablaré a continuación. Haciendo un análisis de su uso, destaca mucho como se puede ver en los siguientes datos:

- Resultados en Google al buscar “Elasticsearch”: 14.300.000
- Seguidores en la red social Twitter (@elastic): 56.113 seguidores
- Fecha de búsqueda: 24 de enero de 2021

Lenovo es una empresa de Fortune 500 con unos ingresos anuales de 45.000 millones de dólares, sin embargo, problemas como escalar de manera exponencial y procesar millones de registros diarios, impidieron a esta empresa desarrollar nuevas iniciativas.

Liuxue Cao, el líder del departamento encargado de esta temática, descubrió que Elasticsearch era la solución a todos los problemas operativos de escalabilidad. En una entrevista dijo: [29] *“Se trata de disminuir la cantidad de esfuerzo. Antes, cuando trabajábamos con bases de datos relacionales tradicionales como MySQL, necesitábamos invertir mucho tiempo optimizando y ajustando para lograr el efecto y el rendimiento que ahora tenemos con Elasticsearch”*.

Una vez que Lenovo hizo la transición a Elasticsearch, obtuvieron el ancho de banda operativo que necesitaban para enfocarse en proyectos que impactaron las decisiones comerciales.

Actualmente Lenovo mantiene 5.000 tiendas minoristas y asociadas en China, las cuales muestran productos de demostración tales como: computadoras, móviles, tabletas y muchos más. Gracias a Elasticsearch consiguieron impulsar las ventas, ya que el equipo de Cao pudo investigar cómo los compradores minoristas interactuaban físicamente con sus dispositivos de hardware. Este proyecto habría sido imposible sin la ayuda de Elasticsearch ya que diariamente se producían alrededor de 400 millones de registros.

Lenovo quería ayudar a los responsables de la toma de decisiones y a los gerentes de las tiendas a comprender las siguientes preguntas: ¿Qué productos y funciones les interesan a los clientes y por qué?, ¿Cómo aumenta el interés la ubicación de la tienda y la ubicación del producto?, ¿Qué productos son de mayor interés en las tiendas minoristas?, ¿El interés del consumidor en los productos Lenovo difiere según la región?

Los datos que responden a estas preguntas se introducen en el *cluster* Elasticsearch de Lenovo. Esta nueva herramienta de métricas, proporciona información en tiempo real sobre la cantidad de tiempo que los consumidores dedican a manejar los productos más nuevos de Lenovo.

Además, los artículos a la venta son colocados estratégicamente dentro de la tienda, para que Lenovo pueda rastrear la popularidad de sus productos en tiempo real y determinar la ubicación óptima de los productos.

Gracias a estas métricas, los equipos minoristas y de ventas son capaces de analizar y realinear su inventario, según sea necesario.

Elasticsearch está impulsando las decisiones de estrategia empresarial en todas las funciones con datos que antes no estaban disponibles debido a la sobrecarga de MySQL de Lenovo. El grupo de ingenieros con el que cuenta, está desarrollando una nueva iniciativa a través de Elasticsearch, para rastrear las ventas de productos correlacionadas con los datos en tiempo real obtenidos en las demostraciones de los productos.

2.1.5 MySQL

2.1.5.1 ¿Qué es MySQL?

Es un sistema de gestión de bases de datos relacionales escrito en C y C++ que fue desarrollado inicialmente por MySQL AB en 1994, posteriormente fue adquirido por Sun Microsystems en 2008 y este a su vez comprado en 2010 por Oracle Corporation. Actualmente, es considerado uno de los *software* de código abierto de este sector más populares del mundo. Puede verse su logotipo en la Figura 2-25 [30].



Figura 2-25 Logotipo MySQL [133]

Las bases de datos de MySQL son relacionales, lo que quiere decir que almacena datos usando una estructura tabular. El modelo lógico con objetos tales como, bases de datos, tablas, vistas, filas y columnas, ofrece un entorno de programación con una gran flexibilidad. Es posible configurar las normas que rigen las relaciones entre los diferentes campos de datos, de forma que, si la base de datos está bien diseñada con estas normas, la aplicación nunca tendrá datos inconscientes, duplicados, huérfanos o desactualizados.

La parte SQL de MySQL significa lenguaje de consulta estructurado (SQL, *Structured Query Language*). Cabe reseñar que es el lenguaje estandarizado más utilizado para acceder a las bases de datos. En función del entorno de programación, pueden darse varios casos, es posible ingresar con SQL directamente, incrustar declaraciones SQL en código escrito en otro lenguaje o usar una API específica del lenguaje que oculta la sintaxis SQL. En la Figura 2-26, se puede observar un ejemplo del lenguaje de MySQL, donde en la tabla “t1”, la entrada de índice para una fila dada, incluye el valor “col1” completo y un prefijo del valor “col2” que consta de sus primeros 10 caracteres.

```
1 CREATE TABLE t1 (  
2   col1 VARCHAR(10),  
3   col2 VARCHAR(20),  
4   INDEX (col1, col2(10))  
5 );
```

Figura 2-26 Ejemplo del lenguaje de MySQL [79]

El servidor de la base de datos MySQL es muy rápido, confiable y escalable. Este software puede ser ejecutado cómodamente desde un ordenador de escritorio o un portátil, incluso permite escalabilidad a grupos de equipos conectados en red. MySQL se desarrolló para manejar grandes bases de datos de manera más rápida y eficiente que las soluciones existentes, y ha sido probado en entornos muy exigentes a lo largo de los años. Gracias a su conectividad, velocidad y seguridad, MySQL es muy útil para acceder a bases de datos en Internet [31].

Las características más relevantes de esta herramienta son [30]:

- Arquitectura cliente/servidor: El funcionamiento de MySQL está basado en un modelo cliente/servidor, de tal manera que estos se comunican entre sí. Cada cliente tiene la

posibilidad de hacer consultas a través del sistema para obtener datos, modificarlos, guardarlos o establecer nuevas tablas de registro.

- Compatibilidad con SQL. Lenguaje de consulta estandarizado por ISO.
- Vista: Es posible personalizarlas, lo cual es un recurso imprescindible en grandes bases de datos.
- Procedimientos almacenados: Se aplican en la manipulación de datos.
- Disparadores: Esta herramienta permite automatizar tareas dentro de la base de datos, de tal forma que, en el momento que se produce un evento, otro es lanzado para actualizar registros u optimizar su funcionalidad.
- Transacciones: El sistema es capaz de preservar la integridad de la base de datos, resguardando la información en caso de que hubiera un fallo.

Como puede verse en la Figura 2-27, MySQL es el segundo sistema de gestión de bases de datos relacionales más popular por detrás de Oracle, según la página web DB-Engines [32]:

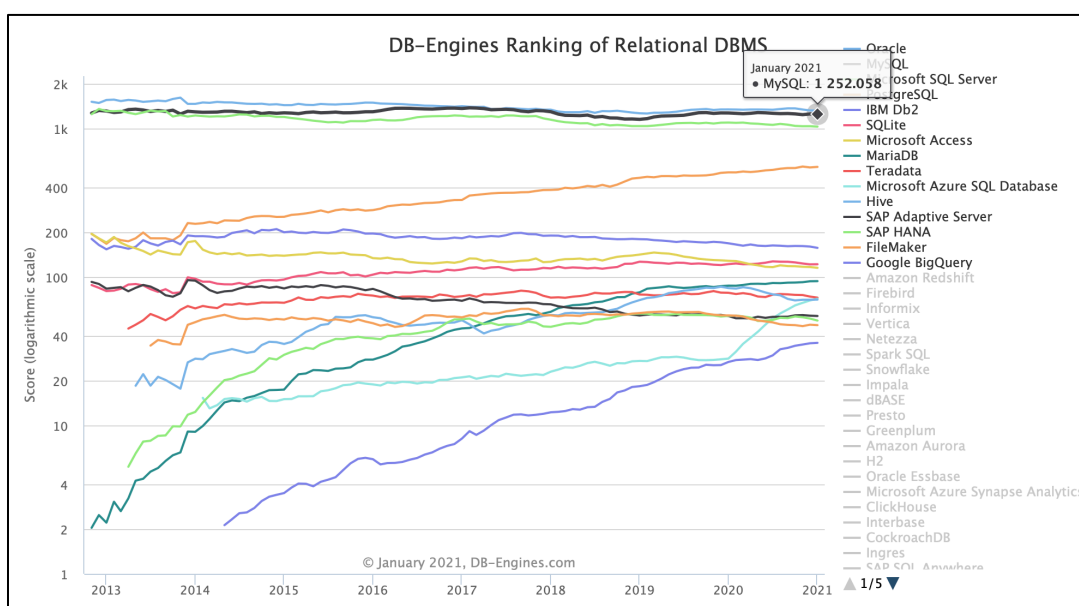


Figura 2-27 Gráfica DBMS de datos relacionales según su popularidad [31].

2.1.5.2 Funcionamiento de MySQL

La característica más notable de la arquitectura de MySQL es que separa el motor de almacenamiento del resto de componentes de la estructura, siendo el encargado de la entrada y salida de información y la representación de esta en la memoria secundaria. Además, su diseño permite cambiar el gestor de almacenamiento para adaptarse al tipo de tarea o aplicación.

MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor de almacenamiento MyISAM, sin embargo, este puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. Por esto, es ideal para aplicaciones web donde la concurrencia en la modificación de los datos sea baja, y el entorno en cuanto a lectura de datos sea exigente.

En la Figura 2-28 se puede ver la arquitectura lógica de MySQL. En esta figura se hace una división entre los componentes que conforman el servidor, las aplicaciones cliente y las partes del sistema operativo en las que se basa el almacenamiento físico [33].

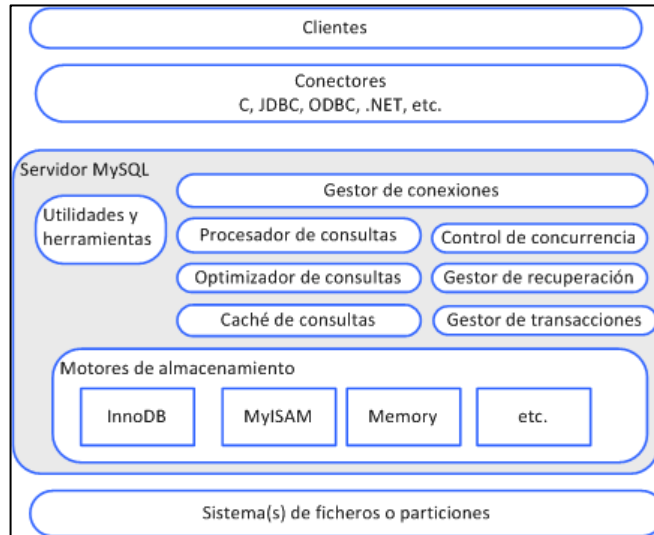


Figura 2-28 Arquitectura lógica de MySQL [33]

A continuación, se va a proceder a explicar cada uno de los componentes más relevantes que forman su arquitectura [33]:

- Los elementos más importantes de esta arquitectura son los motores de almacenamiento, los cuales son reemplazables. La idea de esta estructura es hacer una interfaz abstracta con funciones comunes de gestión de datos a nivel físico. Un motor de almacenamiento puede ser capaz de: tener concurrencia, tener soporte para transacciones, comprobar la integridad referencial, tener un almacenamiento físico, tener soporte de índices y disponer de cachés de memoria. Además, los motores de almacenamiento pueden implementar otras características como la gestión de tipos de datos geoespaciales.
- Los conectores son bibliotecas en diferentes lenguajes de programación que permiten la conexión, ya sea remota o local, con servidores MySQL y la ejecución de consultas.
- El gestor de conexiones es el encargado de mantener las múltiples conexiones de los clientes. Si no hubiera un gestor de conexiones, solo habría una conexión por cada cliente conectado, lo cual sería muy ineficiente. Debe ser capaz de poder configurarse para limitar el número de conexiones concurrentes, ya que las conexiones consumen recursos, y crearlas y destruirlas, son procesos costosos.
- Cada vez que llega una nueva consulta al gestor de MySQL, esta se analiza en el procesador y optimizador de consultas, donde se produce una representación intermedia de la misma. Tras esta representación, MySQL toma una serie de decisiones, que pueden incluir el determinar el orden de lectura de las tablas, el uso de ciertos índices, o la re-escritura de la consulta de manera eficiente.
- En la caché de consultas, se guardan estas y sus resultados. De tal forma, que el procesador de consultas antes de plantearse la optimización, busca la consulta en la caché, reduciéndose así la ineficiencia.
- El control de concurrencia es el mecanismo que se utiliza para evitar efectos no deseados cuando hay lecturas o escrituras simultáneas a la misma porción de datos. Lo que se hace es que cada vez que una aplicación quiera acceder a una porción de los datos, se le proporciona un bloqueo sobre los mismos.
- El gestor de transacciones se encarga de que, si hay algún problema, y parte o alguna de las consultas que forman parte de una transacción, no consigue llevarse a cabo, el servidor anula el efecto parcial de la parte que haya sido ejecutada.

2.1.5.3 Escenario de uso

MySQL es uno de los sistemas de gestión de bases de datos relacionales más populares del mundo, a día de hoy. Haciendo un análisis de su uso, obtenemos:

- Resultados en Google al buscar “MySQL”: 147.000.000
- Seguidores en la red social Twitter (@MySQL): 178.583 seguidores
- Fecha de búsqueda: 25 de enero de 2021

Una de las empresas que optó por esta herramienta para actualizar su infraestructura fue Netflix. El 4 de enero de 2016 en pleno apogeo de la plataforma multimedia Netflix, decidieron migrar su infraestructura de facturación a AWS Cloud (Amazon Web Services) como parte de una iniciativa aún mayor [34].

Para una empresa de esta entidad, su solución de facturación es su sustento financiero y una representación visible de la actitud de la empresa de cara a sus clientes. Teniendo en cuenta que la facturación es un tema sensible por su impacto directo en el bienestar de los clientes, así como en los informes financieros, esta migración debía realizarse de la manera más cuidadosa posible. El objetivo de Netflix era conseguir una migración a la nube por una ruta que fuera segura, resistente y granular, de forma que no afectara a la experiencia de los usuarios.

La infraestructura de facturación es la encargada de administrar el cobro de las suscripciones de Netflix, lo que conlleva realizar un seguimiento de los periodos de facturación abiertos/pagados, controlar la cantidad de crédito en la cuenta del miembro, administrar el estado de pago del miembro, iniciar solicitudes de cargo y la fecha en la que el suscriptor ha pagado, así como otros muchos más datos de facturación. Para poder lograr todo esto, la ingeniería de facturación debía de poder abarcar los siguientes aspectos [34]:

- Trabajos por lotes, para crear pedidos de renovación recurrentes para una base de suscriptores global.
- Datos agregados que se recogen en la Librería General, los cuales se renuevan diariamente, como pueden ser los ingresos de los métodos de pago.
- Servicio de impuestos, el cual va leyendo y publicando las posibles actualizaciones.
- Generación de eventos de mensajería y eventos de *streaming* basados en el estado de facturación de los clientes.
- A través de APIs se brindan detalles de facturación y tarjetas regalo a la plataforma de servicio al cliente y al sitio web. Además, las API de facturación también forman parte de los flujos de trabajo encargados de procesar las acciones del usuario, el registro de miembros, el cambio de plan, la cancelación, la actualización de la dirección, las devoluciones de cargo y las solicitudes de reembolso.
- Integración con otros servicios tales como servicio de cuenta de miembro, procesamiento de pagos, servicio al cliente, mensajería al cliente, etc.

En la arquitectura previa a la migración, los sistemas de facturación tenían elementos tanto en el Netflix Data Center (DC) como en la nube. En la Figura 2-29 se puede ver esta arquitectura a muy alto nivel.

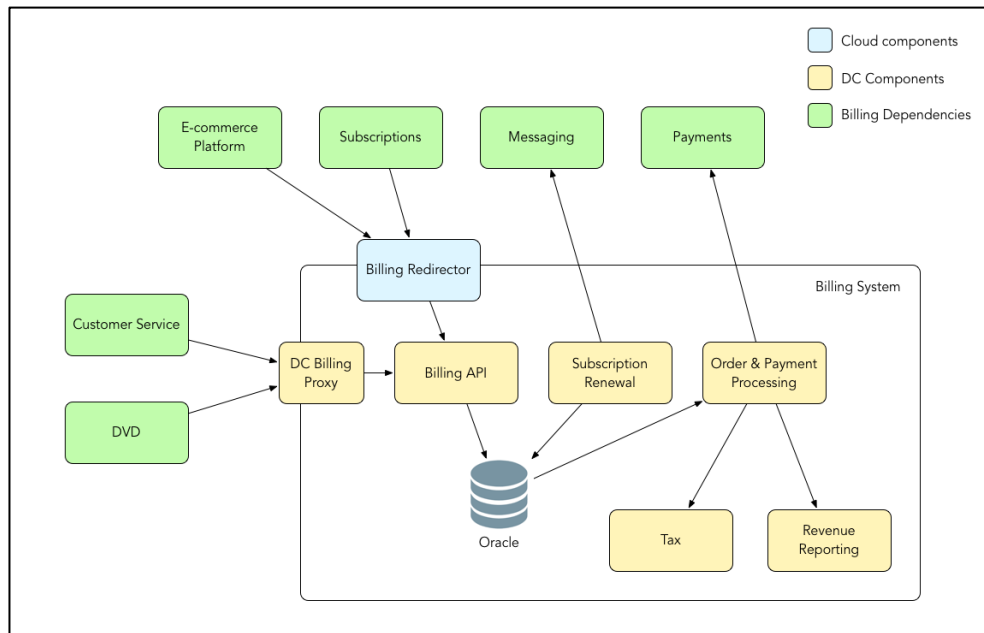


Figura 2-29 Arquitectura Netflix con Oracle [34]

En la arquitectura previa a la migración, la cual estaba basada en Oracle, interactuaban una cantidad de código y datos excesiva, por lo que uno de los objetivos de Netflix llevaba a reducir la complejidad de esa arquitectura. Para ello, algunas de las API debían de ser multirregionales y de alta disponibilidad. La decisión que tomaron fue dividir todos los datos en varios almacenes, de tal forma, que los datos de los suscriptores migraron al almacén de datos de Cassandra, y el resto de datos, migraron a MySQL. En la Figura 2-30 se puede ver como queda la arquitectura tras los nuevos cambios.

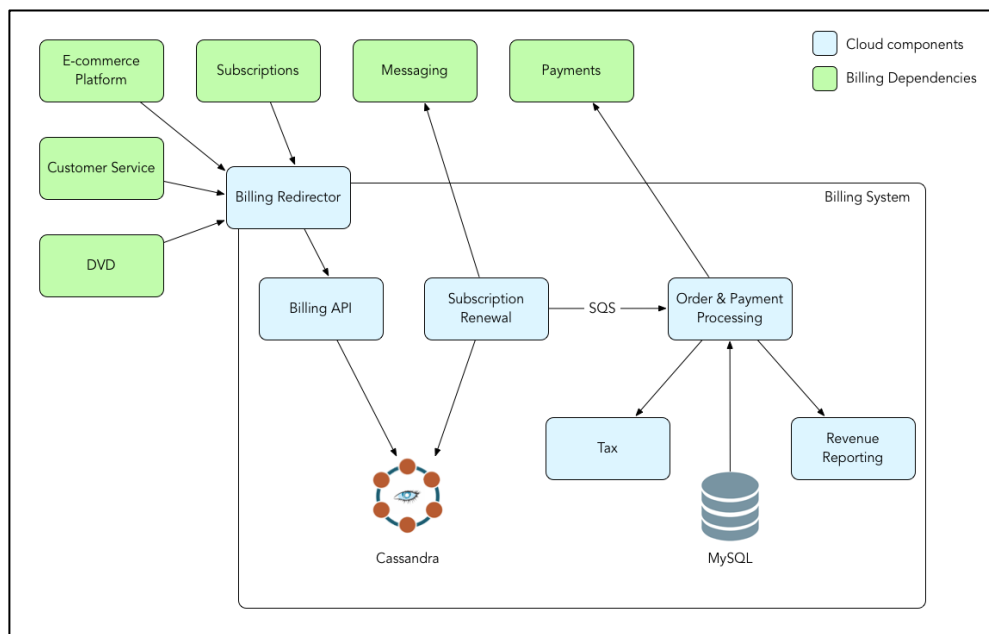


Figura 2-30 Arquitectura Netflix con Cassandra y MySQL [34]

Este cambio no fue fácil, y supuso una serie de desafíos para Netflix, entre los que se encuentran los siguientes [34]:

- La migración no debía causar ningún tiempo de inactividad de cara al usuario.
- La nueva arquitectura en AWS debía permitir una gran escalabilidad para aumentar rápidamente la base de miembros.
- Tenían millones de filas de datos los cuales crecían constantemente, haciendo que aumentara la base de datos Oracle. La cuestión es que, para mover todos esos datos, primero necesitaban transportar y sincronizar los datos en tiempo real.
- El hecho de ser un sistema SOX (Ley “Sarbanes Oxley”) supuso otra dificultad, ya que toda la migración y las herramientas necesarias tenían que adherirse a los procesos SOX. La ley SOX regula las funciones financieras contables y de auditoría.
- Netflix estaba en pleno apogeo y cerca de globalizarse.
- La migración de la facturación debía realizarse sin afectar negativamente a otros equipos que estuvieran ocupados con su propia migración y a los hitos del lanzamiento global.

Netflix decidió migrar de una instalación de Oracle con licencia a una base de datos MySQL de código abierto. Mientras que el procesamiento de suscripciones utilizaba datos en el almacén de datos de Cassandra, el procesador de pagos necesitaba las capacidades de un sistema de gestión de bases de datos relacional para procesar todas las transacciones de cargo. Finalmente, se consiguió definir una arquitectura escalable de múltiples regiones para MySQL con copia DRBD (*Distributed Replicated Block Device*) y múltiples réplicas de lectura, disponibles en diferentes regiones. Además, también se trasladó todo el procesamiento ETL a réplicas para evitar la contención de recursos en MySQL. Se crearon herramientas y alertas para instancias de MySQL para poder garantizar el monitoreo y la recuperación si fuera necesario.

Otro de los grandes desafíos fue migrar datos que estuvieran en constante cambio a MySQL en AWS, sin perder tiempo de inactividad. Tras explorar varias opciones, optaron por Oracle GoldenGate, ya que era capaz de replicar todas las tablas en bases de datos heterogéneas, junto con cambios incrementales continuos. Este movimiento de datos se ejecutó de forma paralela a las operaciones de producción y migración. Se realizaron numerosas pruebas iterativas y ciclos de resolución de problemas para ejecutar las aplicaciones en MySQL. Finalmente, fueron capaces de mover todos sus ETL de procesamiento y datos a MySQL [34].

2.1.6 Apache Kylin

2.1.6.1 ¿Qué es Apache Kylin?

Apache Kylin fue desarrollado por Ebay en el año 2013, y desde 2014 es uno de los proyectos de la “Apache Software Foundation”. En la Figura 2-31 puede verse su logotipo.

Esta herramienta es un almacén de datos analíticos distribuido de código abierto para Big data. Fue diseñado para proporcionar capacidad OLAP (*On-Line Analytical Processing*). Utiliza Hadoop y Spark como motores para el pre-cálculo de cubos multidimensionales y HBase para su almacenamiento. Es capaz de lograr una velocidad de consulta casi constante, independientemente del volumen de datos en constante crecimiento, además de latencias del orden de segundos [35].



Figura 2-31 Logotipo Apache Kylin [35]

Apache Kylin permite consultar miles de millones de filas con latencias inferiores a un segundo siguiendo tres pasos [35]:

- Identificar un esquema en Hadoop.
- Construir un cubo a partir de las tablas identificadas.
- Por último, realizar consultas utilizando ANSI-SQL.

Las funcionalidades y características principales de Apache Kylin son las siguientes [35]:

- Toma de decisiones oportunas sobre Big data. Es capaz de analizar más de 10 mil millones de datos en menos de un segundo.
- Inteligencia empresarial (BI) en Hadoop acelerado.
- Como almacén de datos analíticos, Kylin ofrece ANSI SQL en Hadoop/Spark y soporta la mayoría de las funciones de consulta de ANSI SQL.
- Kylin puede soportar miles de consultas interactivas al mismo tiempo, gracias al bajo consumo de recursos de cada consulta.
- Es posible procesar los datos de *streaming* tan pronto como se generan, permitiendo así el análisis de datos en tiempo real.

Como puede verse en la Figura 2-32, Apache Kylin no tiene mucha popularidad, según la página web DB-Engines [36], ya que se encuentra en el puesto 55 actualmente.

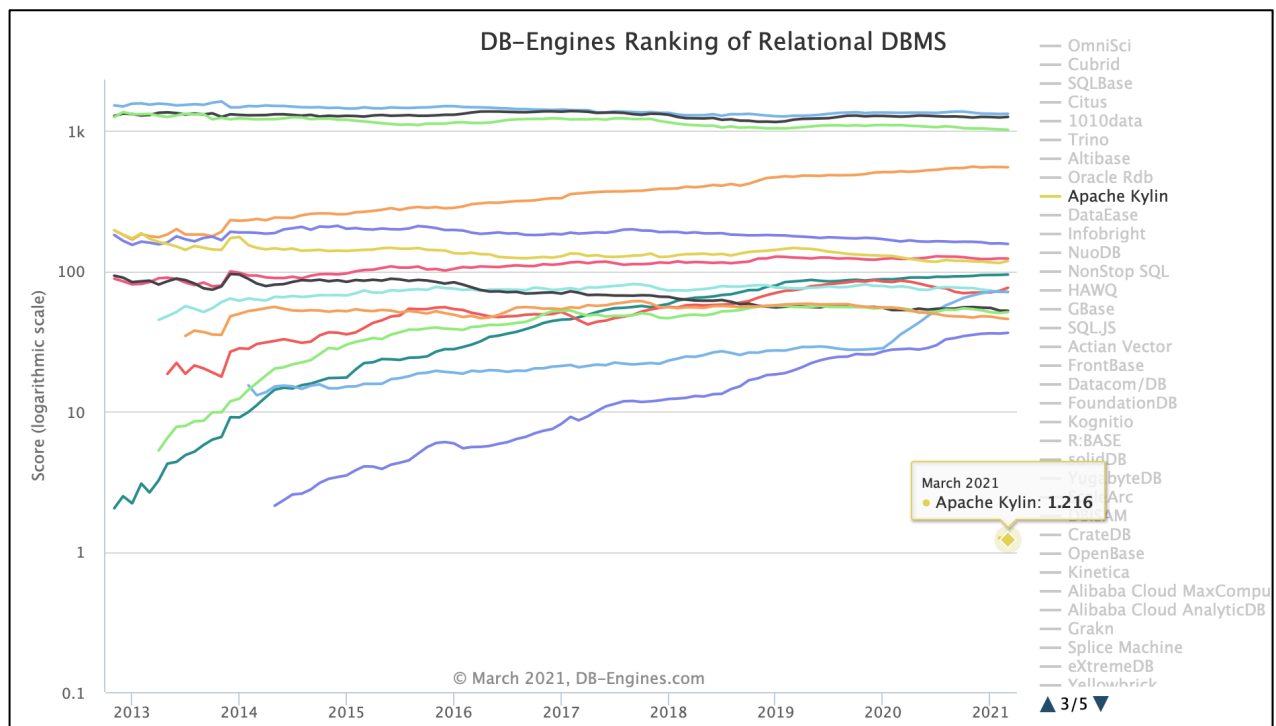


Figura 2-32 Gráfica de las bases de datos relacionales según su popularidad

2.1.6.2 Funcionamiento de Apache Kylin

Para entender el funcionamiento de Apache Kylin conviene saber qué es OLAP; sus siglas quieren decir Procesamiento Analítico Online, y se utiliza para denominar al análisis de datos de registros en varias dimensiones.

El procesamiento OLAP se basa en el modelo de cubos. Un cubo OLAP (véase Figura 2-33) es como un *array* de datos multidimensionales, se caracteriza por contener datos con varias dimensiones o categorías, además de una dimensión temporal. Estas dimensiones o columnas representan los campos por los que se van a agregar los datos.

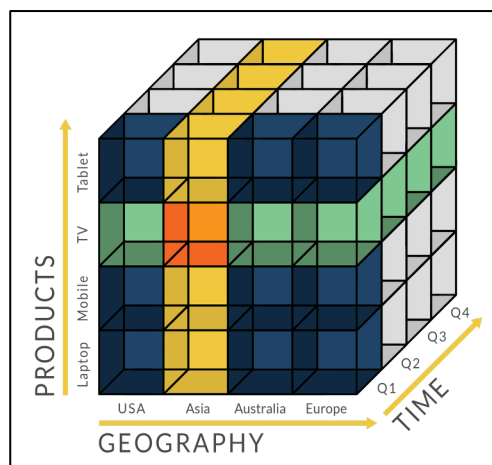


Figura 2-33 Ejemplo de cubo OLAP [124]

Detrás de cada una de las agregaciones hay una función, pudiendo ser un sumatorio, por ejemplo. A su vez, cada agregación se ve reflejada en el cubo.

Una técnica utilizada para acelerar los tiempos de respuesta es pre-calcular las consultas de agregación o agrupación, ya que, de realizar estos cálculos durante el tiempo de la consulta, los tiempos de respuesta aumentarían considerablemente y sería muy ineficiente.

OLAP toma los datos ya presentes en un almacén de datos como Apache Kylin para iniciar los pre-cálculos de los cubos. De tal forma, que el concepto de OLAP en tiempo real consiste en realizar los pre-cálculos a medida que van llegando los datos al sistema. Cada vez que se realiza una de estas operaciones, el resultado está disponible para consulta.

En cuanto a Apache Kylin cabe reseñar que puede soportar varias fuentes de datos, tales como Apache Hive, Apache Kafka, Hadoop y bases de datos relacionales. Además, como motor de procesamiento para llevar a cabo las pre-agregaciones de los cubos OLAP, utiliza Hadoop o Spark.

Para almacenar las pre-agregaciones, utiliza un *cluster* de HBase, garantizando así una latencia de consulta mínima. Los cubos almacenados se parten en segmentos a partir de una columna.

En la Figura 2-34, se puede ver un esquema muy general de la arquitectura de Apache Kylin, donde se observa como entran a jugar los elementos explicados a lo largo del apartado.

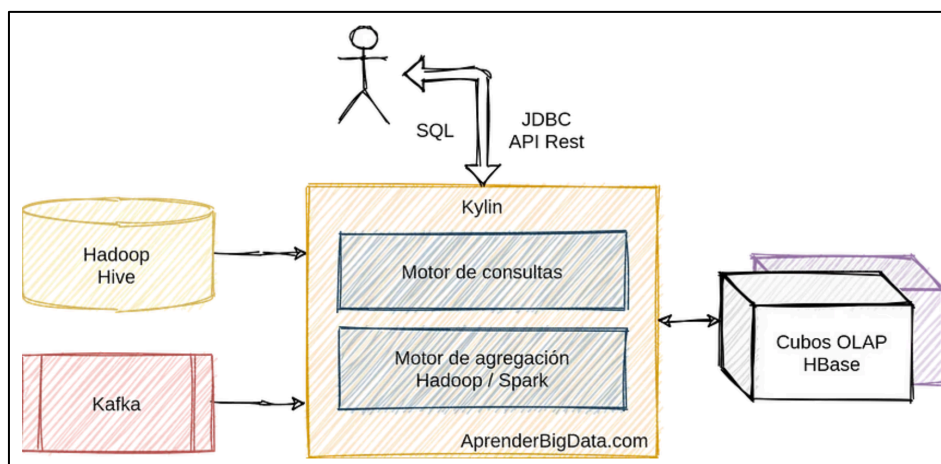


Figura 2-34 Esquema visión general arquitectura Apache Kylin [37]

Transforma los datos de un sistema relacional basado en filas o registros de datos en un sistema de clave-valor que almacena todas las combinaciones requeridas, donde la clave es la combinación de dimensiones agregadas, y el valor es el compuesto de las medidas.

El modelo de datos en Kylin es un diseño en estrella de un cubo, consiste en una tabla de hechos central y varias dimensiones alrededor de ella, las cuales expresan las condiciones de filtro (véase Figura 2-35) [37].

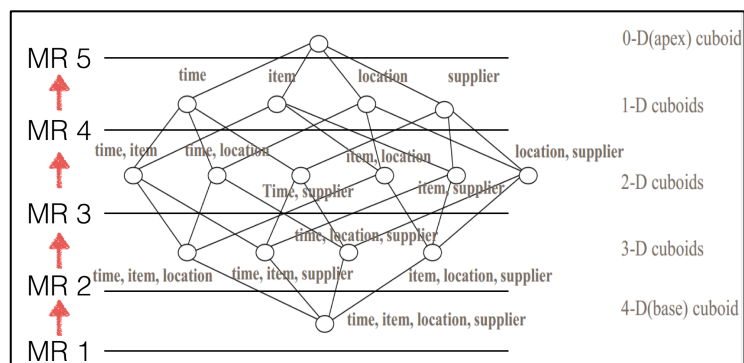


Figura 2-35 Ejemplo cubo OLAP en Apache Kylin [118]

En Kylin conviven dos modelos de procesamiento, los cuales se explican a continuación:

- **Batch OLAP:** Es la primera implementación que se realizó en Kylin. Consiste en los siguientes pasos:
 1. Cargar la tabla de Hive.
 2. Crear el modelo y la descripción del cubo.
 3. Construir el cubo utilizando Apache Spark o Hadoop.
 4. Acceder al cubo con consultas.
- **Real Time OLAP:** En este modelo el objetivo principal es reducir la latencia de los pre-cálculos. De tal forma que, para servir consultas en tiempo real, el servidor de consultas de Kylin examina la *query* para determinar si los datos se encuentran en el histórico (HBase) o, si no han sido pre-calculados aún, se envía la consulta a un nodo *real time*, el cual realiza estas operaciones en memoria, combinando posteriormente los resultados de ambas operaciones.

Como se puede ver en la Figura 2-36, Apache Kylin está formado por los siguientes componentes:

- **REST Server:** Recibe y responde las consultas del usuario o API.
- **Query Engine (Motor de consultas):** Analiza las consultas SQL y se comunica con el motor de almacenamiento.
- **Metadata:** Sistema de gestión persistente de metadatos.
- **Storage Engine (Motor de almacenamiento):** Almacenamiento de cubos subyacente (HBase)
- **Job Engine (Motor de trabajo):** Ejecuta Apache Spark o Hadoop para generar datos de origen en el cubo.

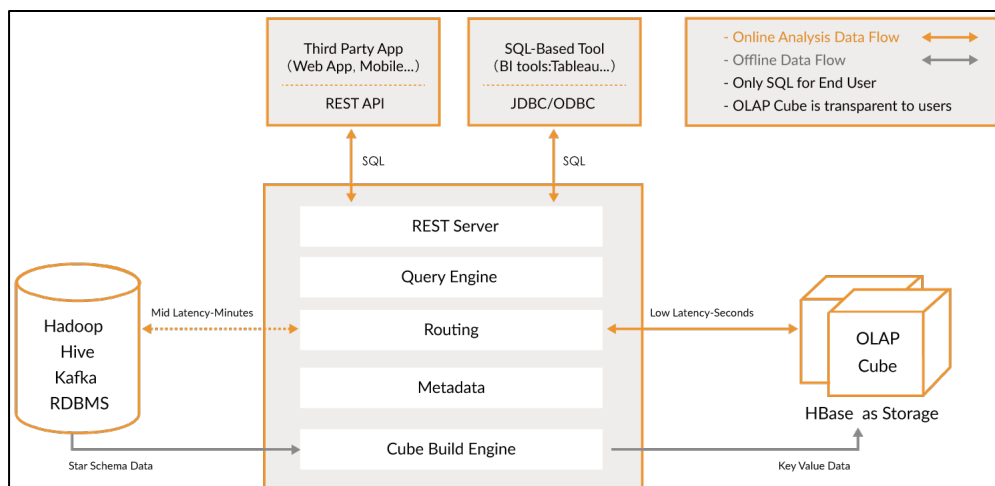


Figura 2-36 Arquitectura Apache Kylin [35] [38]

En la Figura 2-37, se puede observar un ejemplo del lenguaje de consulta utilizado en Apache Kylin, el cual devuelve el promedio de las columnas numéricas.

```
SELECT lstg_format_name, AVG(SUM(price)) OVER(PARTITION BY lstg_format_name) FROM
kylin_sales GROUP BY part_dt, lstg_format_name;
```

Figura 2-37 Ejemplo del lenguaje de consulta de Apache Kylin [59]

2.1.6.3 Escenario de uso

Analizando el uso de Apache Kylin obtenemos los siguientes datos:

- Resultados en Google al buscar “Apache Kylin”: 110.000
- Seguidores en la red social Twitter (@ApacheKylin): 2.963 seguidores
- Fecha de búsqueda: 29 de enero de 2021

Los resultados muestran que no es una herramienta con mucha popularidad a día de hoy, sin embargo, hay algunas empresas como Meituan que han apostado por ella. Meituan, es una plataforma de compras china de consumo local y servicios minoristas que incluyen entretenimiento, comidas, entrega, viajes y otros servicios [38].

A partir de 2015 la empresa comenzó a verse desbordada por el continuo aumento del volumen y la complejidad de datos comerciales. Había principalmente tres problemas, los cuales se detallan a continuación:

- A medida que iban aumentando las dimensiones, el coste de mantener tablas agregadas de varias combinaciones de dimensiones en el almacén de datos iba aumentando cada vez más, reduciéndose así la eficiencia del desarrollo de datos significativamente.
- Con MySQL una vez que alcanzaban los 10 millones de filas, las importaciones y consultas se volvían muy lentas, llegando incluso a bloquearse.
- Debido a la falta de un motor de consulta eficiente para plataformas Big data, todos los requisitos de consulta se ejecutaban en las infraestructuras de almacenamiento de datos (Hive/Presto), lo que hacía que se generase una gran presión informática en el *cluster* y no se pudiera mantener al día el aumento de la demanda empresarial.

Para abordar estos puntos débiles, comenzaron a investigar un motor OLAP más eficiente, buscando soluciones para solidificar los escenarios de consulta.

A principios de 2016 comenzaron a promover soluciones basadas en Kylin en varias líneas comerciales. Tras un año de trabajo, ya habían aplicado Kylin a casi todas las principales líneas de negocio de Meituan. Además, se ha utilizado a gran escala por otras empresas de la misma índole. Kylin se ha convertido en el motor OLAP preferido para estas empresas.

Al poco tiempo, había 214 cubos OLAP en el entorno de producción, el número total de filas de datos incluidas era de 285,3 mil millones y el almacenamiento en HBase era de 59 TB, lo cual satisfacía los requisitos de rendimiento.

Para satisfacer todas sus necesidades, su entorno en línea incluía un *cluster* HBase específico de Kylin de 30 nodos, 2 máquinas físicas para la construcción de cubos y 8 máquinas virtuales utilizadas como máquinas de consulta de Kylin.

Una de las preocupaciones de Meituan era el problema de la “explosión de dimensiones”, es decir, que para cada dimensión adicional, el número de cálculos y almacenamiento del cubo aumentaría de manera exponencial a medida que se duplicara el número de combinaciones de dimensiones. Sin embargo, se vio que en la práctica Kylin no supuso ningún problema, ya que no se requería ningún cálculo previo para todas las combinaciones de dimensiones [38].

2.1.7 Apache Cassandra

2.1.7.1 ¿Qué es Apache Cassandra?

Es una base de datos NoSQL distribuida de código abierto. Tiene un modelo de almacenamiento de columna ancha particionada con semántica consistente.

Se diseñó inicialmente en Facebook, utilizando una arquitectura basada en eventos por etapas, SEDA (*Stayed Event-Drive Architecture*), para poder implementar una combinación de las técnicas de replicación y almacenamiento distribuido Dynamo de Amazon, con el modelo de motor de almacenamiento y datos Bigtable de Google. Tanto Dynamo como Bigtable fueron desarrollados para cumplir con los requisitos emergentes de sistemas de almacenamiento escalables, confiables y de alta disponibilidad, sin embargo, cada uno tenía áreas que podían mejorarse.

Cassandra, cuyo logotipo puede verse en la Figura 2-38, fue diseñada a través de la mejor combinación de ambos sistemas, para poder cumplir con los requisitos de almacenamiento emergentes a gran escala, tanto en la huella de datos como en el volumen de consultas. A medida que las aplicaciones requerían de manera permanente una replicación global completa y escrituras de baja latencia, se hizo necesario diseñar un nuevo tipo de modelo de base de datos, en una época en la que los sistemas de bases de datos relacionales luchaban por cumplir con los nuevos requisitos de las aplicaciones a escala global [39].



Figura 2-38 Logotipo Cassandra [132]

Cassandra es capaz de cumplir esos requerimientos, y tiene por la proa los siguientes desafíos [39]:

- Replicación completa de bases de datos multi-maestro.
- Disponibilidad global con baja latencia.
- Aumentar el rendimiento linealmente con cada procesador adicional.

- Conseguir un equilibrio de carga en línea y crecimiento del *cluster*.
- Consultas orientadas a claves particionadas.
- Tener un esquema flexible.

Debido a la cantidad de datos que se manejan actualmente y, la evolución de estos, ha sido necesario crear una base de datos como Cassandra. Estas bases de datos presentan una mayor flexibilidad a la hora de recoger datos estructurados o semiestructurados. A continuación, se presentan sus características más relevantes [40]:

- Es una base de datos distribuida, lo que permite tener varios servidores operando en paralelo.
- Escala linealmente, lo que significa que cada vez que se duplique el número de nodos, también lo harán las operaciones por cada segundo.
- No sigue un patrón maestro-esclavo, sino que es un *Peer-to-Peer*, lo que conlleva que, si se cae un nodo, el sistema puede seguir funcionando. Al contrario que con el patrón maestro-esclavo, donde si cae el maestro cae todo el sistema.
- Es tolerante a fallos, ya que posee replicación de datos, de tal forma que, si un nodo cae, no pasa nada porque los datos están replicados en otros nodos.
- Permite definir el nivel de consistencia.
- Usa el lenguaje CQL (*Contextual Query Language*). En la Figura 2-39, puede verse un ejemplo de este lenguaje, donde se crea una tabla.

```
CREATE TABLE monkeySpecies (  
    species text PRIMARY KEY,  
    common_name text,  
    population varint,  
    average_size int  
) WITH comment='Important biological records';
```

Figura 2-39 Ejemplo del lenguaje CQL de Apache Cassandra [136]

- Permite la replicación en varios *data center*, siendo cada uno un anillo de máquinas Cassandra. Esto permite que el anillo 1 replique sus datos en el anillo 2, y viceversa.
- Tiene una consistencia ajustable.
- Tiene fácil escalabilidad.

En la Figura 2-40, se puede observar que Apache Cassandra se encuentra actualmente en el primer puesto en el sector de los sistemas de gestión de bases de datos de columna ancha, según la página web BD-Engines [41].

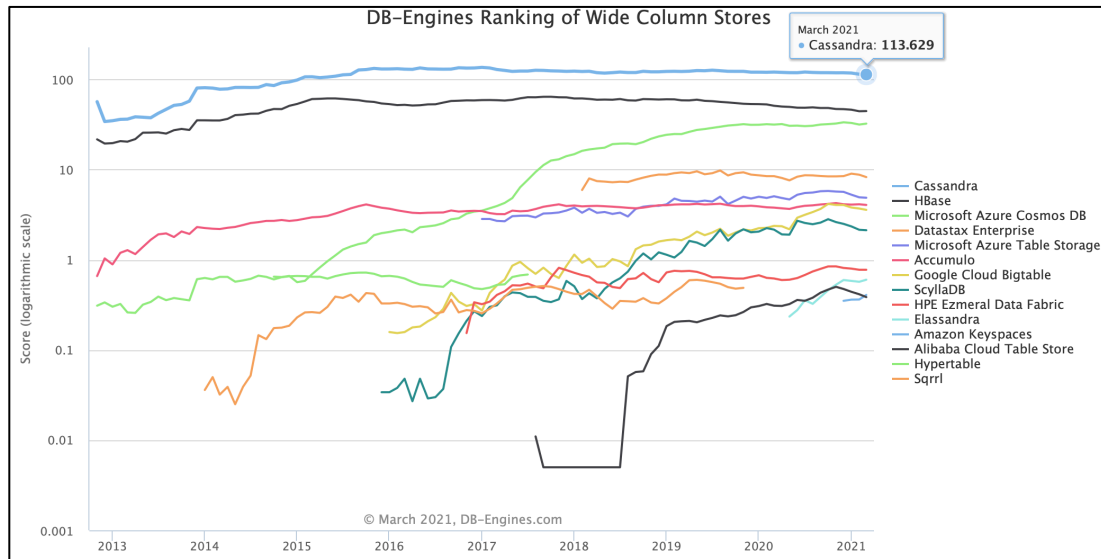


Figura 2-40 Gráfica de los DBMS de columna ancha según su popularidad

2.1.7.2 Funcionamiento de Cassandra

Para comprender el funcionamiento de la base de datos Cassandra es importante conocer primero el Teorema CAP. Este teorema nos dice que un sistema distribuido no puede conseguir al mismo tiempo consistencia, alta disponibilidad y tolerancia a particiones. Es decir, como se puede ver en la Figura 2-41, las bases de datos se van a mover siempre en la intersección de dos de las tres características, pero nunca en el triángulo amarillo, que sería la intersección de las tres. Apache Cassandra estaría situada en la intersección de alta disponibilidad y tolerancia a particiones, sacrificando así un poco de la parte de consistencia [40].

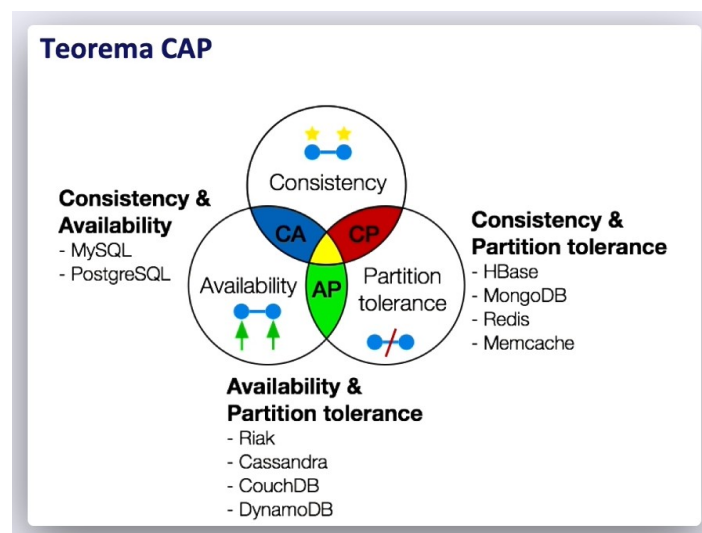


Figura 2-41 Teorema CAP [40]

Cassandra implementa una arquitectura *Peer-to-Peer*, de forma que cualquier nodo puede tomar el rol de coordinador de una *query*. Será el *driver* el que decida qué nodo quiere que sea el coordinador, aunque esta gestión se realiza de manera interna en Cassandra.

Los datos se reparten a lo largo del *cluster* en base a un *token* único calculado para cada fila, mediante una función *hash*, que es un algoritmo matemático que se encargar de transformar cualquier bloque de datos en una nueva serie de caracteres con una longitud fija. Los nodos se reparten de manera

equitativa el rango de *tokens* que va de -2^{63} a 2^{63} , lo que define el nodo primario. Además, Cassandra internamente replica los datos entre los nodos con la política definida por el usuario.

Los elementos que determinan el funcionamiento y la arquitectura de Cassandra, son los siguientes:

- *Cluster*: Se refiere al *data center* (centro de datos).
- *Data center*: Colección de nodos relacionados.
- *Commit Log*: Es el fichero donde se almacena la información sobre los cambios en los datos. Se utiliza para recuperar los datos en caso de que hubiera un fallo en el sistema.
- *MemTable*: Estructura de almacenamiento en memoria. Contiene los datos que aun no han sido escritos en un *SSTable*.
- *SSTable*: Es un fichero que almacenan los datos escritos en disco. Cada fichero *SSTable* es inmutable una vez que se ha creado.
- *Gossip*: Es el protocolo de comunicación *Peer-to-Peer* usado para descubrir y compartir información sobre la localización y estado de los nodos en un *cluster* de Cassandra.
- *Partitioner*: Es un componente que determina cómo se distribuyen los datos entre los nodos.
- *Replica placement strategy*: Define la estrategia para almacenar las copias de los datos en diferentes nodos, asegurando así la accesibilidad y tolerancia a fallos.
- *Snitch*: Define la topología que utilizan las estrategias de replicación, para colocar las réplicas y dirigir las consultas de manera eficiente.

En cuanto al funcionamiento de Cassandra, cabe reseñar el proceso de escritura y lectura. Empezando por el primero, se sigue el siguiente procedimiento:

1. Inicialmente el usuario realiza una petición de escritura, la cual se envía al *MemTable*, donde se almacena la escritura en la memoria RAM.
2. Se realiza la escritura en cada nodo. Además, el *Commit Log* almacena la actividad e información de escritura para poder garantizar la durabilidad de los datos. El *Commit Log* se encuentra en disco.
3. Se produce un *Flush* de la *MemTable*, es decir, que cuando el *MemTable* se queda sin espacio, se eliminan los datos para liberar memoria.
4. Los datos que han sido eliminados se escriben en disco, en las *SSTables*.
5. Una vez que está todo escrito en las *SSTables*, Cassandra trata de reducir de forma periódica el espacio de las *SSTables*, eliminando aquellas que se hayan quedado obsoletas. Los datos actualizados los escribirá en otra *SSTable* y no en la misma (ya que son inmutables). Este proceso recibe el nombre de compactación.

Las escrituras son costosas en el momento en el que la información se escribe en las *SSTables*, sin embargo, si la información está en memoria, en el *MemTable*, su escritura y acceso es mucho más rápida y menos costosa.

Como un recurso auxiliar a la compactación, los datos pueden tener una fecha de caducidad a partir de la cual son marcados por Cassandra como eliminados.

En cuanto al proceso de lectura, puede realizarse de diferentes formas en función de la localización de la información:

- Si los datos se encuentran en la *MemTable*, se procede a la recuperación de ese dato de manera rápida.
- Por otro lado, si los datos no se encuentran en la *MemTable*, se procederá a acceder a las *SSTable* para encontrar esa información.

Además, dentro de estas dos formas de obtener la información, existen tres tipos de lecturas en Cassandra, las cuales se reseñan a continuación:

- *Direct read request*: El nodo coordinador contacta con un nodo que contenga la réplica requerida.
- *Direct request*: El nodo coordinador contacta con tantos nodos con réplicas como se determine en el nivel de consistencia. Se comprueba la consistencia de los datos retornados por el nodo contactado en el *Direct read request*.
- *Background read repair request*: En el caso de existir inconsistencias en las réplicas, las más antiguas son sobrescritas con los datos de las réplicas más actuales [42].

En la Figura 2-42 se puede ver un resumen del proceso de escritura explicado anteriormente.

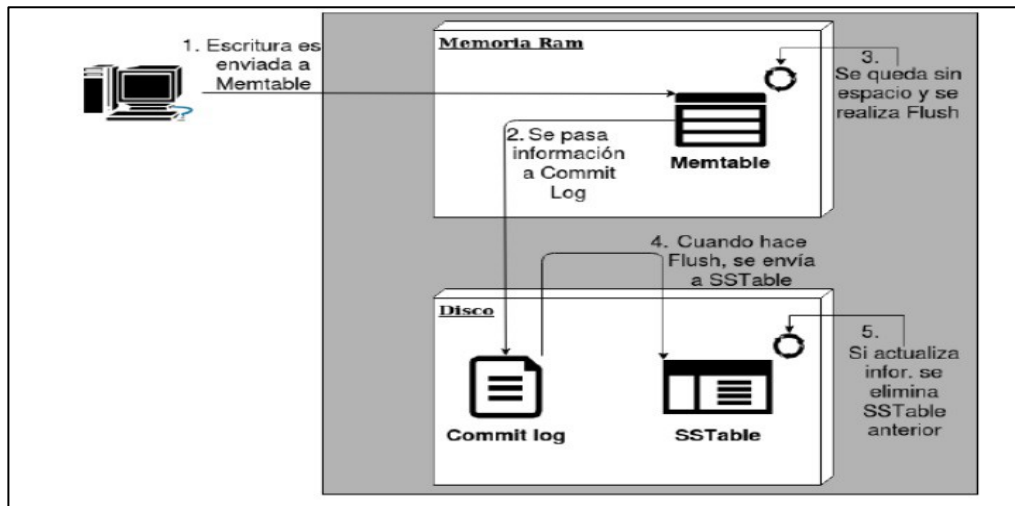


Figura 2-42 Resumen del proceso de escritura [42]

Centrándonos en la arquitectura de Cassandra, En la Figura 2-43 puede verse un ejemplo del *cluster*, de manera muy general.

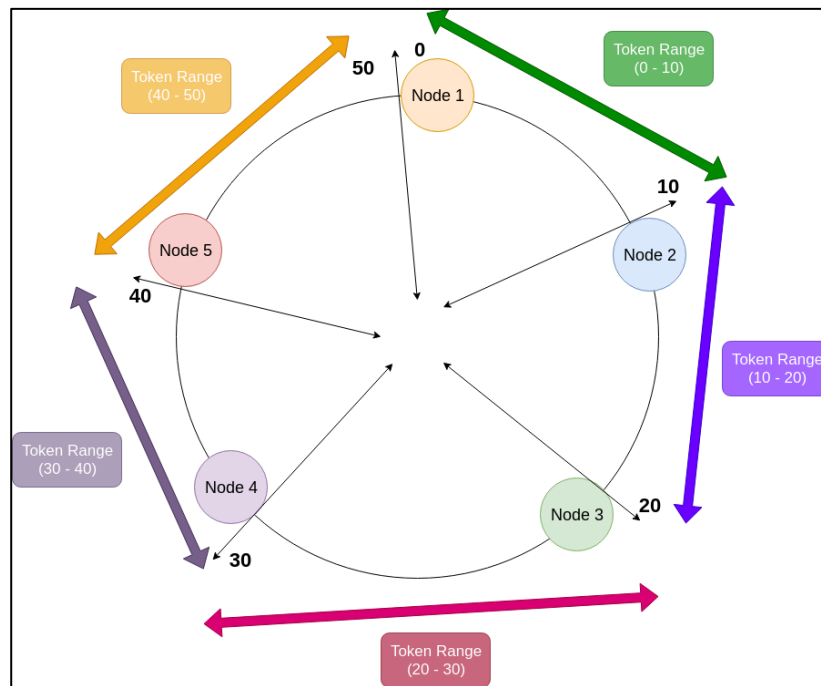


Figura 2-43 Ejemplo estructuración de nodos Cassandra [129]

Los datos del *cluster* se representan como un espacio circular o anillo. Este anillo está dividido en rangos iguales al número de nodos, y además cada nodo es responsable de uno o más rangos de los datos.

Para que un nodo pueda unirse al anillo, se le debe asignar un *token*, el cual determina la posición del nodo en el anillo y el rango de datos del que es responsable. Los datos se dividen en los nodos según la clave de fila. Para que el nodo pueda determinar donde se encuentra la primera réplica de una fila, se recorre el anillo en sentido horario hasta encontrar el nodo que tenga el valor de *token* más alto, siendo cada nodo el responsable de la región del anillo que queda entre él mismo y su predecesor. Una vez que los nodos están ordenados por orden de *tokens*, el último se considera el predecesor del primero, representando así un anillo.

La ubicación de la réplica es calculada mediante la estrategia de colocación de réplica o *Network Topology Strategy*. En cada centro de datos, la primera réplica para una fila particular, está determinada por el valor del *token* asignado a un nodo. Las réplicas adicionales en el mismo centro de datos se van colocando en los nodos en sentido horario hasta llegar al primer nodo, para ello es necesario calcular los *tokens* de partición, quedando los rangos de datos distribuidos uniformemente en cada centro de datos y, evitando así, una distribución desigual de los datos.

2.1.7.3 Escenario de uso

Analizando el uso de Apache Cassandra obtenemos los siguientes resultados:

- Resultados en Google al buscar “Apache Cassandra”: 1.050.000
- Seguidores en la red social Twitter (@cassandra): 40.017 seguidores
- Fecha de búsqueda: 30 de enero de 2021

La red social Instagram, es una de las aplicaciones que hace uso de Apache Cassandra a día de hoy. Cuando se introdujo Instagram en 2010, almacenaban todos sus datos en Cassandra en un solo centro de datos ubicado en los Estados Unidos. Sin embargo, a medida que la empresa comenzó a escalar, se vieron en la necesidad de tener que instalar más centros de datos. De tal forma, que cada vez que añadían un nuevo centro, iban expandiendo la replicación.

A día de hoy, Instagram sirve a millones de usuarios activos en todo el mundo, pero para poder adaptarse a todos esos usuarios y mantener un alto nivel de rendimiento, necesitaba ajustar correctamente su estrategia de ubicación de los centros de datos.

Esta plataforma necesitaba escalar sus recursos de hardware para poder soportar el aumento continuo de la carga de trabajo, lo que se vio reflejado en más centros de datos en más países. Sin embargo, se dieron cuenta de que su estrategia de replicación se quedó corta. Al expandirse a varios países, Instagram se enfrentó a dos desafíos:

- Demasiadas réplicas: Con tres centros de datos, Instagram tenía tres réplicas. Con cinco centros de datos, tenían cinco réplicas, y así sucesivamente. Este problema solo se iría agravando con el tiempo.
- Disminución del rendimiento: las solicitudes de *quórum* implicaban la respuesta de dos réplicas, ubicadas una cerca de la otra. Con el tiempo, las solicitudes de *quórum* requirieron tres réplicas, y cuando provenían de otros continentes, esas réplicas no estaban cerca unas de otras. Esta configuración sub-óptima dañaba el rendimiento.

Cabe reseñar, que el *quórum* es la mayoría de réplicas o también conocido como el factor de replicación, de tal forma que, si tenemos un factor de replicación de 3 en 3 centros de datos, hay 9 réplicas, siendo 5 la mayoría.

Para solucionar los dos problemas mencionados, Instagram decidió dejar de duplicar datos en todas partes. Ya que, al almacenar solo los datos más cercanos a la región donde se generan, la empresa podría construir una red más amplia y eficiente que ofreciera los beneficios de rendimiento que vienen con el acceso a datos locales. Después de todo, la mayoría de usuarios, usan Instagram desde la misma región geográfica la mayor parte del tiempo.

Finalmente, al almacenar datos de EE.UU. en centros de datos con sede en EE.UU. y datos de usuarios europeos en centros de datos europeos, la empresa pudo lograr un rendimiento adecuado [43].

2.1.8 Redis

2.1.8.1 ¿Qué es Redis?

Es un almacén de estructura de datos clave-valor en memoria de código abierto, que fue desarrollado por Salvatore Sanfilippo en 2009. Se utiliza como base de datos, caché y agente de mensajes. Proporciona estructuras de datos tales como cadenas, *hashes*, listas, conjuntos, conjuntos ordenados con consulta de rango, mapas de *bits*, *hiperloglogs*, índices geoespaciales y flujos. Además, tiene replicación incorporada, secuencias de comandos Lua, desalojo de LRU (*Last Recently Used*), transacciones y diferentes niveles de persistencia en disco, y proporciona una alta disponibilidad a través de Redis Sentinel y particionamiento automático con Redis Cluster. Su logotipo puede verse en la Figura 2-44.



Figura 2-44 Logotipo Redis [134]

Redis (*Remote Dictionary Server*) es capaz de ejecutar operaciones tales como, agregar una cadena; incrementar el valor en un *hash*; empujar un elemento a una lista; cómputo de intersección, unión y diferencia de conjuntos; o conseguir al miembro con la clasificación más alta en un conjunto ordenado.

Para poder obtener el máximo rendimiento, trabaja con un conjunto de datos en memoria. Dependiendo del caso de uso, es posible conservar los datos volcando periódicamente el conjunto de datos en el disco o agregando cada comando a un registro basado en disco. También permite deshabilitar la persistencia si solo se necesita un caché en memoria, en red y con muchas funciones.

Redis permite también la replicación asíncrona, con una primera sincronización sin bloqueo muy rápida y reconexión automática con re-sincronización parcial en caso de que se produzca una división de la red.

Se puede utilizar desde la mayoría de los lenguajes de programación. Está escrito en ANSI C y funciona en la mayoría de los sistemas POSIX, como Linux, y OS X, sin dependencias externas. Linux y OS X son los sistemas operativos en los que Redis se desarrolla y se prueba con mayor frecuencia, aunque la página web oficial recomienda Linux, ya que no hay soporte oficial para las compilaciones de Windows [44].

En la Figura 2-45, se puede ver que tiene una gran popularidad dentro del sector de los sistemas de gestión de bases de datos clave-valor, según la página web DB-Engines [45].

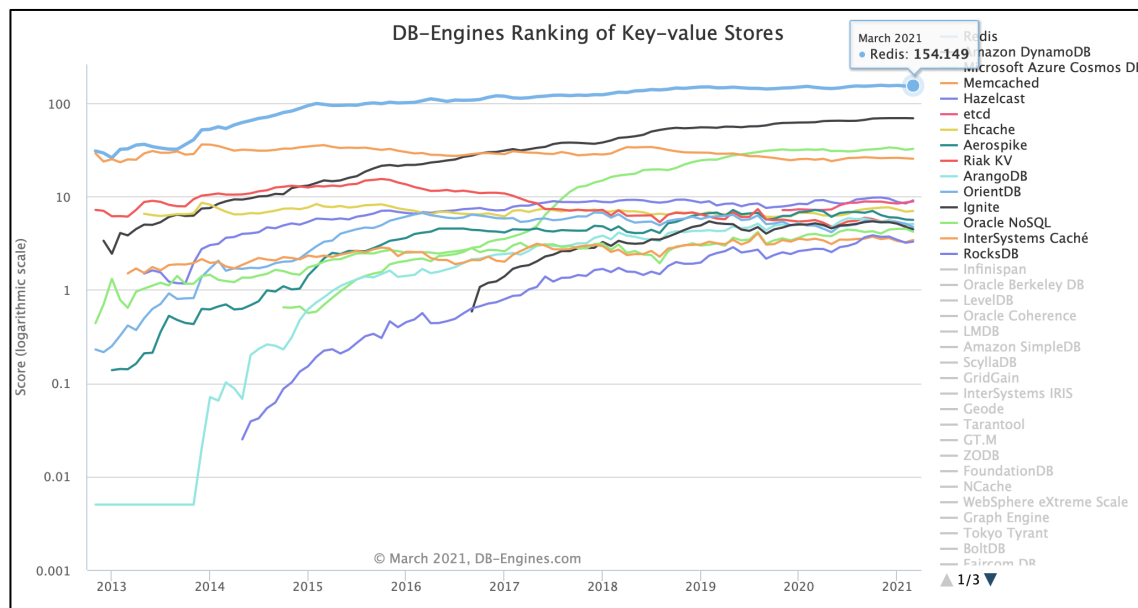


Figura 2-45 Gráfica de los DBSM clave-valor según su popularidad

2.1.8.2 Funcionamiento de Redis

Todos los datos de Redis se encuentran en la memoria principal del servidor, a diferencia de otras bases de datos como Cassandra o MongoDB, que almacenan la mayor parte de los datos en discos o SSD. Mientras que, en las bases de datos tradicionales basadas en disco, la mayoría de operaciones implican ir y volver al disco, los almacenes de datos en memoria no se ven afectados por ello, por lo tanto, permiten mayor cantidad de operaciones y ofrecen tiempos de respuesta más rápidos. El resultado final es un rendimiento extremadamente rápido en las operaciones de lectura o escritura, ejecutándose en menos de un milisegundo y con capacidad para procesar millones de operaciones por segundo.

En la Figura 2-46, puede verse un ejemplo del lenguaje de consulta, donde se puede ver cómo configurar y recuperar un valor de cadena.

```
> set mykey somevalue
OK
> get mykey
"somevalue"
```

Figura 2-46 Ejemplo del lenguaje de consulta de Redis [137]

A diferencia de otros almacenes de datos de clave-valor, los cuales ofrecen estructuras de datos muy simples, Redis cuenta con una gran variedad de estructuras de datos para satisfacer los requisitos de sus aplicaciones. Los tipos de datos incluyen:

- Cadenas: Datos de texto o binarios de hasta 512 MB de tamaño.
- Listas: Colección de cadenas según el orden en el que se van agregando.
- Conjuntos: Una colección desordenada de cadenas con la capacidad para intercalarse, unirse y diferenciarse de otros tipos de conjuntos.
- Conjuntos ordenados: Conjuntos ordenados por un valor.

- *Hashes*: Una estructura de datos para almacenar una lista de pares clave-valor.
- Mapas de *bits*: Un tipo de datos que ofrece operaciones a nivel de bit.
- *HyperLogLogs*: Una estructura de datos probabilísticos para estimar los elementos únicos en un conjunto de datos.

Redis simplifica el código ya que permite escribir menos líneas de código para almacenar, obtener acceso y utilizar datos en las aplicaciones, por ejemplo, si una aplicación tiene datos almacenados en un *hashmap*, es posible hacerlo mediante la estructura de datos *hash* de Redis. En un almacén de datos que no tuviera estructuras de datos *hash* serían necesarias muchas líneas de código para realizar la conversión de un formato a otro.

Esta herramienta utiliza una arquitectura basada en servidor principal y réplica, y además permite la replicación asíncrona, mediante la cual los datos se replican en numerosos servidores de réplicas, logrando un mayor rendimiento de lectura, ya que las solicitudes se reparten entre varios servidores y menores tiempos de recuperación cuando el servidor principal falla. Redis admite copias de seguridad puntuales, copiando el conjunto de datos en el disco, ofreciendo así persistencia.

Esta herramienta nos ofrece un alto nivel de disponibilidad y escalabilidad, gracias a su arquitectura con servidor principal y réplicas en una topología en *clusters* o principal con un único nodo, ofreciendo fiabilidad y rendimiento estables. Cuando es necesario ajustar el tamaño de un *cluster* hay diferentes opciones de escalado [46].

En la Figura 2-47, puede verse un ejemplo de la arquitectura de Redis con replicación asíncrona.

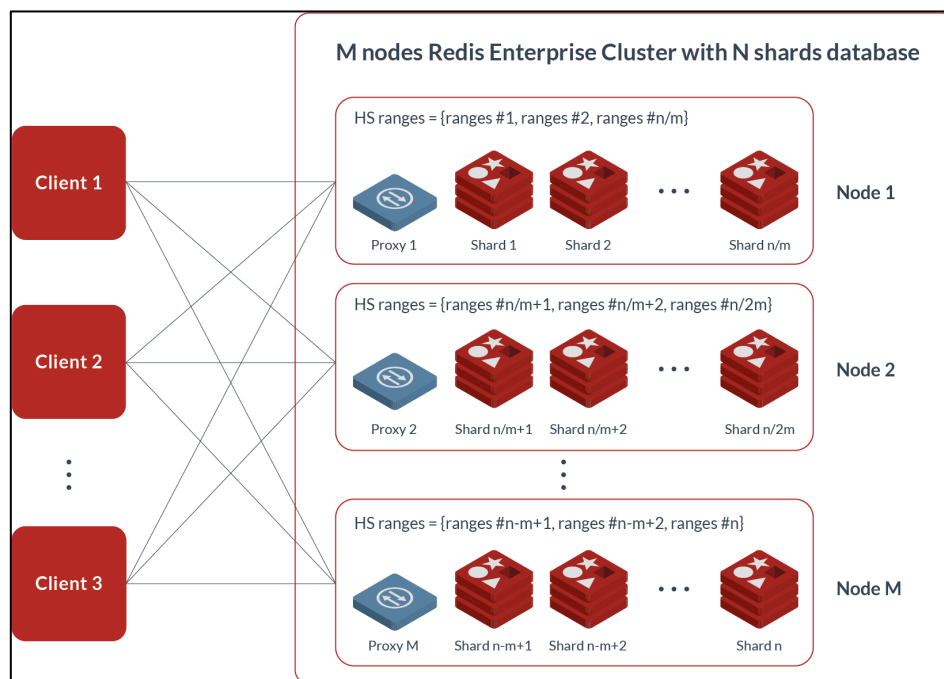


Figura 2-47 Ejemplo de *cluster* con replicación asíncrona [139]

2.1.8.3 Escenario de uso

Realizando un análisis de uso obtenemos los siguientes resultados:

- Resultados en Google al buscar “Redis”: 29.400.000
- Seguidores en la red social Twitter (@RedisLab): 39.764 seguidores
- Fecha de búsqueda: 30 de enero de 2021

Es una herramienta muy popular en varios sectores de la industria, a continuación, se presentan algunos de los principales escenarios de uso [46]:

- **Almacenamiento en caché:** Es una buena opción para implementar una caché en memoria de alta disponibilidad, reduciendo así la latencia de acceso a los datos, incrementando la capacidad de procesamiento y aliviando la carga de la aplicación y de la base de datos relacional o NoSQL. Redis puede suministrar elementos que sean solicitados frecuentemente con tiempos de respuesta inferiores a un milisegundo, además, permite ajustar la escala con facilidad en caso de que la carga aumente, evitando así tener que ampliar el *backend*. El almacenamiento en caché de resultados de consultas en bases de datos, de sesiones persistentes, de páginas web y de objetos de uso frecuente, tales como imágenes, archivos y metadatos, son ejemplos comunes de tipos de almacenamiento en caché que se pueden realizar.
- **Chat, mensajería y colas:** Admite tareas de publicación y suscripción con correspondencia de patrones y una variedad de estructuras de datos, como listas, conjuntos ordenados y *hashes*. Esto permite a Redis respaldar salas de chat de alto rendimiento, *streaming* de comentarios en tiempo real, actividades en redes sociales e intercomunicaciones entre servidores. La estructura de datos de listas facilita la implementación de una cola liviana. Además, las listas ofrecen operaciones atómicas, así como capacidades de bloqueo, lo que hace que resulten muy aptas para una variedad de aplicaciones que requieran un agente de mensajes fiable o una lista circular.
- **Tablas de clasificación de videojuegos:** Los desarrolladores de juegos que necesitan crear tablas de clasificación con datos generados en tiempo real, suelen optar por Redis. La estructura de datos de los conjuntos ordenados de Redis, proporcionan singularidad de elementos a la vez que mantienen la lista ordenada por puntuación de usuarios. Es muy sencillo crear una lista de clasificación en tiempo real, ya que solo es necesario actualizar la puntuación del usuario cuando cambia. También es posible utilizar los conjuntos ordenados para administrar datos de serie temporal con sello de tiempo como puntuación.
- **Almacén de sesiones:** Es una de las opciones favoritas de los desarrolladores de aplicaciones para almacenar y administrar datos de sesiones para aplicaciones a escala de Internet. Redis ofrece la latencia, escalabilidad y resiliencia necesarias para administrar datos de sesiones, como perfiles de usuarios, credenciales, estados de sesiones y personalización específica para usuarios.
- **Streaming completo de contenido multimedia:** Ofrece un almacén de datos en memoria ágil para casos de uso de *streaming* en directo. El almacenamiento de metadatos de Redis se puede usar para perfiles de usuarios e historial de visualizaciones, *tokens*/información de autenticación para millones de usuarios y archivos de manifiestos para permitir hacer *streaming* de vídeos a millones de usuarios de aplicaciones móviles y de escritorio en un determinado momento.
- **Análisis geoespacial:** Ofrece estructuras de datos en memoria y operadores personalizados para administrar datos geoespaciales a escala y con velocidad. Los comandos como *Geoadd*, *Geodist*, *Georadius* y *Georadiusbymember* son utilizados para almacenar, procesar y analizar datos geoespaciales en tiempo real, facilitando y agilizando las tareas con Redis y agregando a las aplicaciones características basadas en ubicación geográfica, tales como tiempo de conducción, distancia recorrida y puntos de interés.
- **Machine Learning:** Redis proporciona un almacén de datos en memoria para crear, entrenar e implementar rápidamente modelos de *Machine Learning*. Las aplicaciones modernas basadas en datos exigen que el *Machine Learning* procese rápidamente grandes volúmenes de datos variados y ágiles, y que automatice la toma de decisiones. Para casos de uso como la detección de fraudes en juego y servicios financieros, las subastas en tiempo real en el sector de la tecnología publicitaria y las coincidencias en aplicaciones de citas y viajes compartidos, la capacidad para procesar datos instantáneamente y tomar decisiones en decenas de milisegundos, tiene una importancia fundamental.

- **Análisis en tiempo real:** Redis se puede usar con soluciones de *streaming*, como Apache Kafka, como almacén de datos en memoria para incorporar, procesar y analizar datos en tiempo real con una latencia inferior a un milisegundo. Es por ello, que es la opción ideal para casos de uso de análisis en tiempo real, como análisis de datos en redes sociales, focalización de anuncios, personalización e IoT.

2.1.9 Comparación de las herramientas

En este apartado se procede a comparar las herramientas para determinar cuál es la más conveniente para este proyecto. Para ello, se ha hecho una tabla incluyendo las funcionalidades que deberían tener las herramientas para cumplir con los requisitos necesarios:

- **Modelo de base de datos:** Es importante reseñarlo, ya que cada modelo puede ser útil para una aplicación en concreto. Las herramientas estudiadas presentan los siguientes modelos:
 - **Base de datos documental:** Son muy flexibles, ya que la estructura de los documentos no tiene por qué ser coherente, permitiendo almacenar conjuntos de datos estructurados muy grandes en una sola base de datos. Además, es relativamente sencillo integrar datos nuevos, ya que basta con integrar la información nueva en solo unos pocos atributos, al contrario que las bases de datos relacionales, donde la nueva información debe insertarse en todos los atributos. En estas bases de datos, además, la información no se reparte en tablas, sino que se almacena en el mismo lugar, lo que puede favorecer al rendimiento. Sin embargo, cuando se intentan conectar datos entre sí, el sistema se vuelve muy complejo y voluminoso [47].
 - **Base de datos de series temporales:** Se utilizan para el almacenamiento de series temporales y consultas en tiempo real. Además, están optimizadas para una gran velocidad de lectura y escritura en tiempo real, así como la realización de operaciones de filtrado y agregación basadas en los sellos temporales. Al ser optimizadas para un tipo de datos específico, suelen ocupar menos espacio que las bases de datos NoSQL. Suelen ser fácilmente asociables con un software de visualización. La desventaja es que no soportan datos no estructurados [48].
 - **Motor de búsqueda:** Permiten trabajar con un gran volumen de datos de cualquier tipo y acceder a ellos en tiempo real. Además, es posible operar sobre distintos índices al mismo tiempo para potenciar las búsquedas. Los datos se almacenan en diferentes sistemas que colaboran entre sí y muestran los resultados requeridos en una sola petición [49].
 - **Base de datos relacional:** Almacenan sus datos en tablas formadas por filas y columnas, permitiendo manejar grandes cantidades de datos relacionados entre sí, de manera sencilla, conforme a unas normas y de modo uniforme. Mantienen la uniformidad de los datos en todas las aplicaciones y copias de la propia base, denominadas instancias, garantizando así que todas las copias de las bases de datos tienen los mismos datos, y también que no se produzcan duplicidades de registros. Además, para evitar conflictos cuando varios usuarios o aplicaciones intentan acceder a los mismos datos al mismo tiempo, se bloquea dicho acceso mientras los datos se estén actualizando. Sin embargo, también tienen una serie de desventajas, ya que son deficientes a la hora de manejar datos de grafos, multimedia y sistemas de información geográfica; tampoco permiten crear sub-filas, estando todas las filas al mismo nivel jerárquico. El rendimiento puede no ser muy bueno a la hora de hacer consultas y obtener resultados con gran cantidad de tablas [50].
 - **Almacén de columna ancha:** Utilizan tablas, columnas y filas, pero a diferencia de las bases de datos relacionales, el formato y el nombre de las columnas pueden variar de una fila a otra dentro de la misma tabla. Además, cada columna es almacenada por separado en el disco, por lo que requieren poco almacenamiento y realizan las

consultas de manera rápida y eficiente, gracias a que permiten una comprensión de los atributos. Cada uno de los atributos se almacena por separado en bloques, por lo que el proceso de toma de decisiones es más rápido, al igual que las combinaciones. Otra ventaja es que es más sencillo agregar nuevas columnas a la base de datos, cada vez que hay nuevos atributos estas bases de datos simplemente crean otro archivo nuevo para la columna nueva, evitando así reconstruir tablas enormes. Como desventaja, las actualizaciones pueden ser ineficaces por la gran cantidad de bloques [51].

- Cubos OLAP: Permiten procesar grandes volúmenes de datos en campos bien definidos y con un acceso inmediato a los datos para su consulta, sin embargo, tienen la desventaja de que no es posible realizar cambios en su estructura, sino que habría que rediseñar el cubo OLAP [52].
- Bases de datos clave-valor: Son muy efectivas a la hora de realizar consultas y muy escalables, gracias a su sencilla estructura. El almacenamiento clave-valor no exige ningún esquema fijo por lo que es posible realizar modificaciones sobre la marcha en la base de datos, añadiendo nuevos campos mientras se realizan acciones en otras entradas. Además, también ofrecen una gran velocidad de búsqueda gracias a la sencilla conexión entre la clave y el valor. Una desventaja de este método, es que no se contempla otro método de acceso, ya que únicamente está prevista la búsqueda a través de la clave [53].
- Creación de índices: Es otra de las funcionalidades requeridas en las herramientas, la posibilidad de crear índices definidos por el usuario, permitiendo una clasificación de los datos y con ello, un rápido acceso a estos en las consultas.
- Consultas geoespaciales: En este proyecto es importante esta funcionalidad, ya que puede sernos de gran utilidad para, por ejemplo, buscar barcos que estén en un área definida por unas coordenadas.
- Consultas temporales: Resulta muy útil para este proyecto poder consultar los datos que hay en un intervalo de tiempo definido, por ejemplo, saber las posiciones de un barco en las últimas seis horas.
- Operaciones matemáticas: Es relevante estudiar si las herramientas pueden realizar operaciones matemáticas básicas, tales como sumas y restas, para, por ejemplo, calcular cuántos eventos AIS ha emitido un barco. Por ello es necesario poder hacer una consulta en la que sea posible sumar el número de eventos AIS de un barco.
- Interacción con *dashboards*: Es reseñable estudiar si tienen posible interacción con *dashboards* de uso popular como Grafana y Kibana.
- Interacción con Apache Storm como motor analítico.
- Popularidad: La opinión de los usuarios y la acogida que tienen las herramientas en la comunidad del sector, es relativamente importante para saber si la herramienta es eficaz y fiable a primera vista. Además, la popularidad de la herramienta suele estar correlacionada con la existencia de documentación, tutoriales o foros en los que resolver dudas sobre la herramienta. Es por ello que ha sido uno de los criterios implementados en la tabla. Para determinar la popularidad, se ha hecho a partir del número de seguidores en la red social Twitter y de los resultados de búsqueda en Google. Para ello, se han definido tres niveles: alto, medio y bajo. Siendo: alto, una cantidad superior a 50.000 seguidores en Twitter; medio, entre 20.000 y 50.000; y bajo, por debajo de 20.000. En cuanto a resultados de Google, el criterio ha sido el siguiente: Alto, por encima de 5 millones de resultados; medio, entre 1 y 5 millones; y bajo, por debajo de 1 millón.

En la Tabla 4 se puede ver la comparación entre las herramientas estudiadas, según las funcionalidades de interés explicadas anteriormente

	MongoDb	InfluxDB	KsqlDB	Elasticsearch	MySQL	Apache Kylin	Apache Cassandra	Redis
Modelo de base de datos	Almacén de documentos/motor de búsqueda	Series temporales	Series temporales	Motor de búsqueda/Almacén de documentos	Relacional	OLAP	Almacén de columna ancha	Almacén clave-valor
Lenguaje de consulta	MQL [54]	Flux [55]	SQL [56]	Query DSL [57]	SQL [58]	SQL [59]	CQL [60]	NoSQL [61]
Interacción Dashboards (Kibana, Grafana)	Sí [62]	Sí [63]	No, se hace a través de Kafka [64]	Sí [65]	Sí [66]	No, tiene su propio <i>dashboard</i> [35]	Sí [67]	Sí [68]
Interacción Apache Storm	Sí [69]	Sí [70]	No, integración con Kafka	Sí [71]	Sí [72]	No	Sí [73]	Sí [74]
Creación de índices (definidos por el usuario)	Sí [75]	Sí [76]	Sí [77]	Sí [78]	Sí [79]	No, cubos OLAP [80]	Sí [81]	No [82]
Consultas temporales	Sí [83]	Sí [84]	Sí [85]	Sí [86]	Sí [87] [88]	Sí [89]	Sí [90]	Sí [91]
Permite búsqueda geoespacial	Sí [92]	Sí [93]	No, con ciertas particularidades [94] [95] [96]	Sí [97]	No, se necesita una implementación [98]	No [99]	Sí [100]	Sí, pero no permite polígonos irregulares [101]
Soporta operaciones matemáticas	Sí [102]	Sí [103]	Sí [104]	Sí [105]	Sí [106]	Sí [107]	Sí [108]	Sí [109]
Popularidad	Alta	Media	Baja	Alta	Alta	Baja	Media	Alta

Tabla 4 Tabla comparativa de las herramientas

Tras hacer una comparativa entre las diferentes herramientas, es posible empezar a descartar aquellas que tengan baja popularidad, descartamos así KsqlDB y Apache Kylin.

Redis no permite la creación de índices ni búsquedas geoespaciales en polígonos irregulares, y MySQL necesita una implementación para poder realizar consultas geoespaciales, además de que no tiene muy buena escalabilidad, por lo que podemos descartar ambas.

Podemos desechar también aquellas herramientas cuyos modelos de base de datos no sean acordes a los requerimientos del proyecto, ya que se pretende trabajar con flujos de datos en tiempo real, o al menos, simular como si lo fuera, es por ello que podríamos descartar MongoDB y Apache Cassandra.

De esta forma nos quedamos finalmente con InfluxDB y Elasticsearch, ya que ambas permiten trabajar con series temporales.

Para tomar una decisión entre ambas herramientas, se plantean una serie de funcionalidades básicas y necesarias para este proyecto, como son:

- Consultas tipo *look up*: Por ejemplo, buscar barcos a través de su MMSI y que nos devuelva datos de interés de este.
- Consultas temporales: Como puede ser, saber las últimas posiciones de un barco en las últimas 10 horas.
- Consultas geoespaciales: Por ejemplo, buscar barcos en el interior de un polígono definido por una serie de coordenadas.
- Permitir rotar los datos, es decir, que haya una zona caliente y una fría, de tal forma que podamos pedirle a la herramienta que mantenga en la zona caliente los datos de las últimas horas, para que sea más fácil acceder a ellos.

Sin embargo, ambas herramientas ofrecen este tipo de funcionalidades y consultas, por lo que habrá que decidirse por una de ellas evaluando su rendimiento. Para ello se ha utilizado el estudio realizado en la referencia [110].

La comparación se hará en función de los siguientes factores:

- Rendimiento en la ingesta de datos.
- Requisitos de almacenamiento en disco.
- Tiempo medio de respuesta a la consulta.

Para las pruebas se utilizaron 45 millones de entradas. Los resultados son los siguientes:

	InfluxDB	Elasticsearch
Rendimiento en la ingesta de datos	• 15m 20s • 50000 pts/s	• 1h 24m 8s • 8000 docs/s
Almacenamiento en disco	356 MB	1,6 GB
Tiempo medio de respuesta de la consulta	• 100 consultas: 2m 35s • Una consulta: ○ Sin caché: 1,5s	• 100 consultas: 12,6s • Una consulta: ○ Sin caché: 130ms ○ Con caché: 15 ms

Tabla 5 Comparación rendimiento InfluxDB y Elasticsearch

InfluxDB ofrece un mejor rendimiento en cuanto a la ingesta de datos (5,6 veces mejor que Elasticsearch) y requiere menos espacio en disco. Sin embargo, para realizar consultas complejas lleva bastante tiempo. En cambio, Elasticsearch tiene un tiempo de respuesta mucho menor, aunque requiere mayor espacio en disco y es más lento en la indexación.

Además, cabe reseñar que ambas herramientas tienen buena escalabilidad, pero Elasticsearch lo hace con la versión de código libre, mientras que la agrupación de clústeres de InfluxDB está en la edición comercial.

Es por ello que finalmente, Elasticsearch va a ser la herramienta utilizada para este proyecto.

3 DESARROLLO DEL TFG

Tras una comparación cualitativa de las diferentes herramientas descritas en el Capítulo 2 y la elección final de Elasticsearch para este proyecto, en este capítulo se explica de manera detallada el proceso de instalación, así como algunas funcionalidades de esta herramienta.

3.1 Proceso de instalación Elasticsearch

3.1.1 Características del equipo.

Antes de comenzar, se muestran a continuación las especificaciones técnicas del equipo en el que se van a llevar a cabo tanto la instalación, como las pruebas de funcionamiento que se detallan en este capítulo. El equipo es un MacBook Air (2018), con las siguientes características:

- Sistema operativo: macOS Big Sur (Versión 11.0.1).
- Procesador: 1,6 GHz Intel Core i5 de doble núcleo.
- Memoria: 8 GB 2133 MHz LPDDR3.
- Gráficos: Intel UHD Graphics 617 1536 MB.

Todo el proceso de instalación y funcionamiento, se ha llevado a cabo en una máquina virtual de Ubuntu 20.04.

3.1.2 Proceso de instalación de Elasticsearch.

Toda la explicación de las diferentes consultas reflejadas en los siguientes apartados ha sido obtenida de la referencia [26]. La herramienta ofrece muchas más funcionalidades, pero se ha querido explicar aquellas que cobran mayor importancia en este proyecto. En la página web oficial de Elasticsearch [111] descargamos el archivo de instalación. La última versión es la 7.10.2 y data del 14 de enero de 2021. En el caso de este proyecto, el formato a descargar ha sido para el sistema operativo Linux, sin embargo, la página web ofrece una gran variedad de formatos para diferentes sistemas operativos, tales como macOS o Windows. El archivo descargado viene nombrado de la siguiente manera: *elasticsearch-7.10.2-linux-x86_64.tar.gz*

Es recomendable mover el archivo a un directorio accesible fácilmente para trabajar, en este caso, el directorio elegido es */Documents*. Una vez en el directorio deseado, procedemos a la descompresión del archivo haciendo doble clic sobre el mismo. En la Figura 3-1 se puede observar el contenido de la carpeta tras la descompresión del archivo descargado.



Figura 3-1 Contenido de la carpeta Elasticsearch-7.10.2

Tras esto, abrimos el terminal para llevar a cabo la instalación de Elasticsearch desde el directorio `/bin`. Comenzarán a instalarse todos los paquetes necesarios, tras lo cual es recomendable hacer un test para ver si se ha instalado correctamente. Debería obtenerse una respuesta como la que se ve en la Figura 3-2, lo que quiere decir que el *cluster* de Elasticsearch está en funcionamiento y listo para empezar a experimentar con él. Los comandos a ejecutar son:

```
-$ cd Documentos/elasticsearch-7.10.2
-$ bin./elasticsearch
-$ curl 'http://localhost:9200/?pretty'
```

```
javier@javier-VirtualBox:~$ curl 'http://localhost:9200/?pretty'
{
  "name" : "javier-VirtualBox",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "xpTMlo-5TzKEA5CV11CbuA",
  "version" : {
    "number" : "7.11.0",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "8ced7813d6f16d2ef30792e2fcde3e755795ee04",
    "build_date" : "2021-02-08T22:44:01.320463Z",
    "build_snapshot" : false,
    "lucene_version" : "8.7.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figura 3-2 Test de funcionamiento de Elasticsearch

A continuación, conviene cambiar el nombre que viene por defecto del *cluster* (*cluster.name*). En este caso le he puesto AIS Data. Para modificar el nombre, hay que editar el archivo *elasticsearch.yml* que se encuentra en el directorio `/config`, tras esto es necesario reiniciar Elasticsearch. Cabe reseñar que para detener Elasticsearch cuando está funcionando, es posible hacerlo presionando las teclas Ctrl+C.

Usando la *cat health API* es posible verificar si el *cluster* está funcionando correctamente. La respuesta nos debería indicar que el estado del *cluster* de Elasticsearch es *green*, como se puede ver en la Figura 3-3.

```
-$ curl -X GET "localhost:9200/_cat/health?v=true&pretty"
```

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/_cat/health?v=true&pretty"
epoch      timestamp cluster      status node.total node.data shards pri relo i
nit unassign pending_tasks max_task_wait_time active_shards_percent
1613074365 20:12:45 elasticsearch green          1          1      0 0 0
0          0          0          -          100.0%
```

Figura 3-3 Test de funcionamiento del *cluster* de Elasticsearch mediante *cat health API*

3.1.3 Funcionalidades básicas.

En este apartado se explicarán las funcionalidades más útiles que ofrece Elasticsearch para este proyecto, como: crear índices, indexar documentos, realizar consultas, agregaciones, flujos de datos y consultas geográficas.

3.1.3.1 Crear índice

Para agregar datos en Elasticsearch, es necesario un índice, un lugar para almacenar datos relacionados, no es más que un espacio de nombres lógico que apunta a uno o más *shards* físicos.

Un *shard* es una unidad de trabajo de bajo nivel, la cual contiene una sola parte de todos los datos del índice. Los documentos son almacenados e indexados en *shards*, sin embargo, las aplicaciones no se comunican con ellos directamente, sino que hablan con el índice.

Son la forma en que Elasticsearch distribuye los datos en el *cluster*, pudiendo compararse a contenedores de datos. Los documentos se almacenan en *shards*, y estos a su vez se asignan a los nodos del *cluster*. A medida que este va creciendo o reduciéndose, Elasticsearch va migrando automáticamente los *shards* entre los nodos, para que el *cluster* permanezca equilibrado en todo momento.

Un *shard* puede ser primario o réplica. Cada documento del índice pertenece a un solo *shard* primario, de tal forma que la cantidad de *shards* primarios determina la cantidad máxima de datos que puede contener el índice.

Un *shard* réplica, no es más que una copia del *shard* principal. Las réplicas se utilizan para proporcionar copias redundantes de los datos, protegiéndolos así contra fallos del hardware, y también para atender a solicitudes de lectura, tales como buscar o recuperar un documento.

En el momento de crear el índice el número de *shards* primarios queda fijado y no es posible cambiarlo, sin embargo, el número de réplicas sí es posible modificarlo en cualquier momento. En la Figura 3-4 se puede observar el comando utilizado para crear un nuevo índice, con tres *shards* primarios y una réplica por cada uno de ellos.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/example1?pretty" -H 'Content-Type: application/json' -d'
{
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 1
    }
  },
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "example1"
}
```

Figura 3-4 Creación de un índice

Ahora mismo, las réplicas no tendrían sentido ya que solo hay un nodo activo, por lo que sería recomendable iniciar otro nodo para evitar así tener un único punto de fallo, sin redundancia. En el ejemplo de la figura anterior, se ha designado una réplica por cada *shard* primario, de tal forma que, al iniciar un nuevo nodo, las réplicas pasarían a este y los *shards* primarios se quedarían en el primer nodo, proporcionando así redundancia en los datos.

Cabe destacar que también es posible eliminar los índices creados, tal y como se ve en la Figura 3-5.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X DELETE "localhost:9200/example1?pretty"
{
  "acknowledged" : true
}
```

Figura 3-5 Borrar un índice

3.1.3.2 Indexar documentos

Una vez que el *cluster* esté en funcionamiento, ya es posible indexar algunos datos, para ello lo que se hace es almacenar documentos JSON en un índice de Elasticsearch. Esto es posible hacerlo directamente a través de una solicitud *PUT*, donde se especifica el índice al que se desea agregar el documento, un ID de documento único y uno o más pares de clave-valor en el cuerpo de la solicitud. Como se puede ver en la Figura 3-6, esta solicitud crea el índice “customer”, agregando un nuevo documento en este, con ID 1, y, además, almacena e indexa el campo de nombre.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/customer/_doc/1?pretty" -H 'Content-Type: application/json' -d'
> {
>   "name": "Javier Vila"
> }
>
>
{
  "_index" : "customer",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 3,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 2,
  "_primary_term" : 2
}
```

Figura 3-6 Solicitud PUT

El nuevo documento creado, está disponible inmediatamente desde cualquier nodo del *cluster*. Es posible recuperarlo, a través de la solicitud GET que especifica el ID del documento. Como se puede ver en la Figura 3-7, la respuesta nos muestra el documento con el ID especificado y muestra los campos que se indexaron anteriormente con la solicitud PUT.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/customer/_doc/1?pretty"
{
  "_index" : "customer",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 3,
  "_seq_no" : 2,
  "_primary_term" : 2,
  "found" : true,
  "_source" : {
    "name" : "Javier Vila"
  }
}
```

Figura 3-7 Solicitud GET

La realidad es que Elasticsearch es utilizado para poder indexar una gran cantidad de documentos, estos pueden incluirse en lotes a través de la *bulk* API. Este método es mucho más rápido que ir enviando solicitudes individuales, ya que se minimizan los viajes de ida y vuelta en la red.

La página web oficial nos ofrece un fichero a modo de ejemplo [112], (*accounts.json*) el cual incluye un conjunto de datos generado aleatoriamente, los cuales representan cuentas bancarias (véase Figura 3-8).

```
{"index":{"_id":"1"}}
{"account_number":1,"balance":39225,"firstname":"Amber","lastname":"Duke","age":32,"gender":"M","addresses":"880 Holmes Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}
```

Figura 3-8 Ejemplo de cuentas bancarias del fichero *accounts.json*

A través de una solicitud *bulk*, indexamos los datos de las cuentas que se incluyen en el fichero. Como se puede ver en la Figura 3-9, hay 1.000 documentos indexados correctamente.

```
-$ curl -H "Content-Type application/json" -XPOST "localhost:9200/bank/_bulk?pretty&refresh"
  -data-binary "@accounts.json
-$ curl "localhost:9200/_cat/indices?v=true"
```

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl "localhost:9200/_cat/indices?v=true"
health status index      uuid                                pri rep docs.count docs.deleted store.size pri.store.size
yellow open   bank      d0Dez460Q0GqQR5fQRFKoQ           1   1     1000           0     411.3kb       411.3kb
yellow open   customer D6p4DLfeSC26Lx10IE6u1g           1   1         1           0        3.8kb         3.8kb
```

Figura 3-9 Documentos indexados

3.1.3.3 Solicitudes de búsqueda.

Tras haber indexado los datos de las cuentas que se incluyen en el fichero *accounts.json*, Elasticsearch permite hacer solicitudes de búsqueda. Para acceder al conjunto de capacidades de búsqueda, hay que usar Elasticsearch Query DSL (*Domain Specific Language*) para especificar los criterios de búsqueda en el cuerpo de la solicitud, además, también hay que indicar el nombre del índice en el que se desea buscar. En la Figura 3-10, puede verse un ejemplo de solicitud de búsqueda, donde se pide que nos devuelva los datos del fichero ordenados por números de cuenta de manera ascendente, para ello se utiliza el método GET.

La respuesta nos proporciona la siguiente información acerca de la solicitud de búsqueda:

- *took*: El tiempo en milisegundos que le ha llevado a Elasticsearch hacer la búsqueda.
- *timed_out*: Indica si se ha agotado o no el tiempo de espera de la solicitud de búsqueda.
- *_shards*: En cuantos *shards* se ha realizado la búsqueda, y un desglose de cuantos *shards* tuvieron éxito, fallaron o se omitieron.
- *max_score*: la puntuación del documento más relevante.
- *hits.total.value*: Cuantos documentos coincidentes se han encontrado.
- *hits.sort*: La posición de clasificación del documento (cuando no se clasifica por puntuación de relevancia).
- *hits._score*: La puntuación de relevancia del documento (no se aplica cuando se usa *match_all*)

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": { "match_all": {} },
>   "sort": [
>     { "account_number": "asc" }
>   ]
> }
> '
{
  "took" : 301,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "_doc",
        "_id" : "0",
        "_score" : null,
        "_source" : {
          "account_number" : 0,
          "balance" : 16623,
          "firstname" : "Bradshaw",
          "lastname" : "Mckenzie",
          "age" : 29,
          "gender" : "F",
          "address" : "244 Columbus Place",
          "employer" : "Euron",
          "email" : "bradshawmckenzie@euron.com",
          "city" : "Hobucken",
          "state" : "CO"
        },
        "sort" : [
          0
        ]
      },

```

Figura 3-10 Solicitud de búsqueda en el fichero *accounts.json*

En Elasticsearch cada solicitud es autónoma, de tal forma que no mantiene ninguna información de estado entre las solicitudes. Es posible desplazarse por los resultados de la búsqueda especificando en la solicitud los parámetros de origen y el tamaño, por ejemplo, si en una solicitud se especifican tanto el origen (*from*) como el tamaño (*size*) en 10, obtenemos los resultados del 10 al 19.

Tras saber cómo hacer una solicitud de búsqueda básica, el siguiente paso es realizar consultas más personalizadas que con *match_all*. Si el objetivo es buscar datos específicos dentro de un campo, es posible realizar una consulta que busque coincidencias, como en la Figura 3-11, que se realiza una solicitud para buscar direcciones que contengan los términos *mill lane* en el campo de dirección del fichero *accounts.json*.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": { "match": { "address": "mill lane" } }
> }
> '
{
  "took" : 123,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 19,
      "relation" : "eq"
    },
    "max_score" : 9.507477,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "_doc",
        "_id" : "136",
        "_score" : 9.507477,
        "_source" : {
          "account_number" : 136,
          "balance" : 45801,
          "firstname" : "Winnie",
          "lastname" : "Holland",
          "age" : 38,
          "gender" : "M",
          "address" : "198 Mill Lane",
          "employer" : "Neteria",
          "email" : "winnieholland@neteria.com",
          "city" : "Urie",
          "state" : "IL"
        }
      }
    ]
  }
},
```

Figura 3-11 Solicitud de búsqueda de una dirección en concreto

Además, para realizar una búsqueda de una frase en lugar de términos individuales como la anterior, se puede utilizar el comando *match_phrase* en lugar de *match*. Un ejemplo sería, realizar la misma búsqueda de antes, pero en vez de solicitando todas aquellas direcciones que contengan cualquiera de las dos palabras (*mill* o *lane*), solicitar únicamente aquellas respuestas que contengan la frase “*mill lane*”.

Es posible combinar varios criterios de consulta a través de consultas *bool*. Elasticsearch permite designar criterios como *must* (obligatorios), *should* (deseables) o *must_not* (no deseables). En la Figura 3-12, se puede ver un ejemplo de este tipo de solicitud, donde se busca en el índice bancario cuentas que pertenezcan a clientes que tengan 40 años, pero excluye a cualquier persona que viva en Idaho (ID).

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": {
>     "bool": {
>       "must": [
>         { "match": { "age": "40" } }
>       ],
>       "must_not": [
>         { "match": { "state": "ID" } }
>       ]
>     }
>   }
> }
> .
{
  "took" : 46,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 43,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "_doc",
        "_id" : "474",
        "_score" : 1.0,
        "_source" : {
          "account_number" : 474,
          "balance" : 35896,
          "firstname" : "O'Brien",
          "lastname" : "Walton",
          "age" : 40,
          "gender" : "F",
          "address" : "192 Ide Court",
          "employer" : "Suremax",
          "email" : "obrienwalton@suremax.com",
          "city" : "Crucible",
          "state" : "UT"
        }
      }
    ]
  }
}
```

Figura 3-12 Solicitud de búsqueda combinando varios criterios

Cada elemento *must*, *should* y *must_no* de una consulta booleana, recibe el nombre de cláusula de consulta. La puntuación de relevancia del documento se define en función del grado en que un documento cumple los criterios de cada cláusula *must* o *should*, de tal forma que, cuanto mayor sea la puntuación, mejor coincidirá el documento con sus criterios de búsqueda. Elasticsearch devuelve los documentos clasificados según estas puntuaciones de relevancia de manera predeterminada.

Los criterios de una cláusula *must_not*, se tratan como un filtro, afectando a si el documento se incluye o no en los resultados, pero no contribuye a cómo se puntúan los documentos. Es posible también especificar explícitamente filtros arbitrarios para incluir o excluir documentos basados en datos estructurados. En la Figura 3-13 se puede observar un ejemplo en el cual la solicitud usa un filtro de rango para limitar los resultados a las cuentas con un saldo entre \$20.000 y \$30.000.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": {
>     "bool": {
>       "must": { "match_all": {} },
>       "filter": {
>         "range": {
>           "balance": {
>             "gte": 20000,
>             "lte": 30000
>           }
>         }
>       }
>     }
>   }
> }
```

Figura 3-13 Solicitud de búsqueda con filtro de rango

Elasticsearch permite realizar también consultas para recuperar documentos que contengan un prefijo específico en un campo proporcionado. En la Figura 3-14, puede verse un ejemplo de búsqueda en el que se solicitan los documentos del campo “user.id” que contengan un término que comience por “ki”.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": {
>     "prefix": {
>       "user.id": {
>         "value": "ki"
>       }
>     }
>   }
> }
```

Figura 3-14 Búsqueda de prefijo

También es posible recuperar los campos que nosotros queramos en el momento de la consulta. De forma predeterminada, en Elasticsearch cada resultado de la respuesta de búsqueda incluye el documento *_source*, que es el objeto JSON completo que se proporcionó al indexar el documento. Sin embargo, para recuperar campos específicos en la respuesta de búsqueda, se puede usar el parámetro *fields*, tal y como se ve en la Figura 3-15, donde se solicita únicamente el campo “mmsi” en la respuesta.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/aisdata/_search?pretty" -H 'Content-Type: application/json' -o Barcos_Algeciras.txt -d'
{
  "query": {
    "match": {
      "micro": "ALGECIRAS"
    }
  },
  "fields": ["mmsi"], "size": 10000,
  "_source": false
}'
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  9267  100  9162  100  105  79669    913  --:--:-- --:--:-- --:--:--  79887
```

Figura 3-15 Restricción de campos "fields" en la consulta

El parámetro *fields* consulta tanto el *_source* de un documento, como el *mapping* (mapeo) de un índice para cargar y devolver los valores. *Mapping* es el proceso de definir cómo se almacena e indexa un documento y los campos que contiene.

Otra opción interesante de las búsquedas en Elasticsearch, es que permite ordenar los resultados de búsqueda a través de *sort*. Es posible añadir uno o más *sorts* en campos específicos, siendo posible ordenarlos según *score* (*_score*) o según el orden del índice (*_doc*), para ello se puede utilizar *asc* (*sort* en orden ascendente) o *desc* (*sort* en orden descendente).

Cuando se ordena según el *score*, el orden predeterminado es *desc*, sin embargo, para ordenar cualquier otra cosa, el valor predeterminado es *asc*. En la Figura 3-16, se puede observar el uso de *sort* en una consulta para ordenar las últimas 20 posiciones de un barco, de tal manera que se obtengan primero las más recientes.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/aisdata/_search?pretty" -H 'Content-Type: application/json' -o 20_Ultimas_posiciones.txt -d'
{
  "query": {
    "match": {
      "mmsi": "351983000"
    }
  }, "sort": [{"@timestamp": {"order": "desc"}}],
  "fields": ["position"], "size": 20,
  "_source": false
}'
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  1281  100  1134  100  147  28350   3675  --:--:-- --:--:-- --:--:--  32025
```

Figura 3-16 Uso de "sort" para ordenar los resultados de la búsqueda

3.1.3.4 Agregaciones

En Elasticsearch las agregaciones permiten obtener metainformación sobre los resultados de búsqueda y responder preguntas tales como, “¿Cuál es el saldo promedio de cuentas en Tennessee?” o “¿Cuántos titulares de cuentas hay en Texas?”. Es posible buscar documentos, filtrar los resultados y, además, usar agregaciones para analizar los resultados en una sola solicitud. En la Figura 3-17, se puede ver una solicitud en la que se utiliza una agregación, para agrupar todas las cuentas por estados en el índice bancario y, además, nos devuelve los diez estados con la mayor cantidad de cuentas en orden descendente.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "size": 0,
>   "aggs": {
>     "group_by_state": {
>       "terms": {
>         "field": "state.keyword"
>       }
>     }
>   }
> }
> '
{
  "took" : 1175,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_state" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 743,
      "buckets" : [
        {
          "key" : "TX",
          "doc_count" : 30
        },
        {
          "key" : "MD",
          "doc_count" : 28
        },
        {
          "key" : "ID",
          "doc_count" : 27
        }
      ]
    }
  }
}
```

Figura 3-17 Solicitud de búsqueda con agregación

Los *buckets* en la respuesta son los valores del campo referido a los estados. El *doc_count* nos indica el número de cuentas bancarias en cada estado, en el caso de la Figura 3-18 vemos que en Texas (ID) hay 30 cuentas de usuario. Además, debido a que en la solicitud se indica *size=0*, la respuesta solo contiene los resultados de la agregación.

También es posible combinar agregaciones para crear resúmenes más complejos de los datos. Como se puede observar en la Figura 3-18, la siguiente solicitud indica una agregación *avg* (promedio), de la agregación anterior (*group_by_state*), para calcular los saldos de cuenta promedio en cada estado.

Se puede especificar el orden en el que queremos que aparezcan los resultados de la agregación, dentro de los *terms* (términos) de la agregación, indicándolo a través de “*order*”, en lugar de ordenar los resultados por recuento como en el ejemplo anterior.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword"
      },
      "aggs": {
        "average_balance": {
          "avg": {
            "field": "balance"
          }
        }
      }
    }
  }
}
'
{
  "took" : 179,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_state" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 743,
      "buckets" : [
        {
          "key" : "TX",
          "doc_count" : 30,
          "average_balance" : {
            "value" : 26073.3
          }
        }
      ]
    }
  }
}
```

Figura 3-18 Solicitud de búsqueda con combinación de agregaciones

Además de este *bucketing* básico y agregaciones de métricas como las de los ejemplos anteriores. Elasticsearch proporciona también agregaciones especializadas para operar en múltiples campos y analizar datos como fechas, direcciones IP y datos geográficos. Las capacidades de análisis proporcionadas por las agregaciones habilitan funciones avanzadas como el uso de aprendizaje automático para detectar anomalías.

3.1.4 Flujos de datos

Un flujo de datos permite almacenar datos de series temporales en múltiples índices, con un solo recurso con nombre para las solicitudes, siendo adecuados para registros, eventos, métricas y otros datos que se generen de manera continua.

Es posible enviar solicitudes de indexación y búsqueda directamente a un flujo de datos, de tal forma que la solicitud se enruta automáticamente a índices de respaldo, que almacenan los datos de la secuencia. Los índices de respaldo se pueden automatizar mediante la gestión del ciclo de vida del índice (ILM). Por ejemplo, ILM podría usarse para:

- Hacer girar un nuevo índice cuando un índice alcance un cierto tamaño o número de documentos.
- Crear un nuevo índice cada día, semana o mes y archivar los anteriores.
- Eliminar índices que se hayan quedado obsoletos.

Un flujo de datos consiste en un índice de respaldo o más, los cuales se generan automáticamente (véase Figura 3-19).

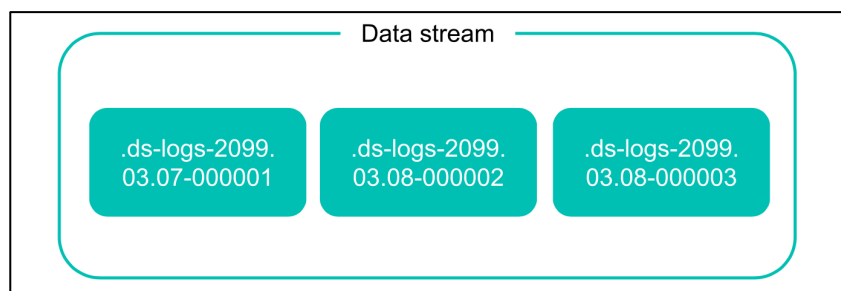


Figura 3-19 Índices de respaldo

Cada flujo requiere una plantilla de índice (*index template*) coincidente, la cual contiene las asignaciones y configuraciones que se utilizan para configurar los índices de respaldo de la secuencia.

Los documentos indexados deben contener un campo *@timestamp*, mapeado como un tipo de campo *date* o *date_nanos*. Si la plantilla de índice no incluyera un *@timestamp*, Elasticsearch se encarga de asignar de manera predeterminada un *@timestamp* como un campo de fecha.

La misma plantilla de índice puede ser utilizada para múltiples flujos de datos. Cabe reseñar que no puede ser eliminada una plantilla de índice que este en uso por un flujo de datos.

Cuando una solicitud de lectura es enviada, el flujo enruta la solicitud a todos los índices de respaldo, como se puede ver en el esquema de la Figura 3-20.

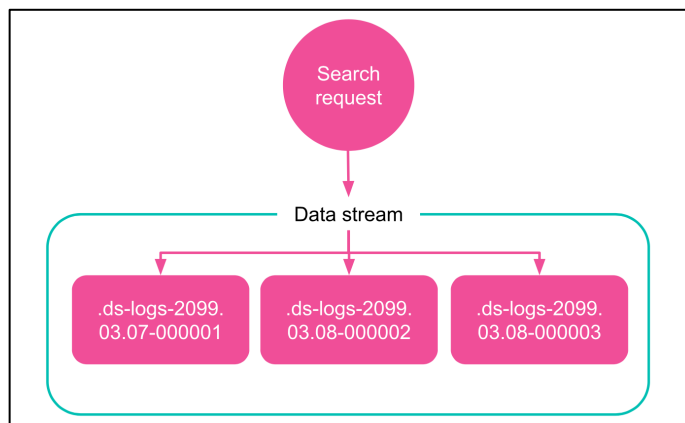


Figura 3-20 Solicitud de lectura enviada a los índices de respaldo

El índice de respaldo que se haya creado más recientemente es el índice de escritura, de tal forma que el flujo solo agrega nuevos documentos a este índice, como se puede ver en la Figura 3-21. No es posible agregar nuevos documentos a otros índices de respaldo, incluso enviando solicitudes directamente al índice.

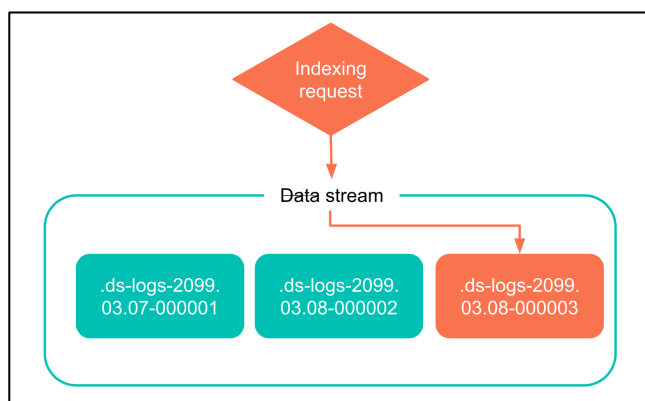


Figura 3-21 Índice de escritura

Un *rollover* crea un nuevo índice de respaldo, el cual se convierte en el nuevo índice de escritura de la secuencia, siendo posible automatizarlo a través de ILM. Elasticsearch al indexar continuamente documentos con marca temporal, normalmente utiliza un flujo de datos para que pueda pasar periódicamente a un nuevo índice, permitiendo implementar una arquitectura *hot-warm-cold* para cumplir con los requisitos de rendimiento en los datos más recientes, controlar los costes a lo largo del tiempo y hacer cumplir las políticas de retención.

Para poder automatizar la renovación y la gestión de un flujo de datos con ILM, hay que seguir los siguientes pasos:

- Crear una política del ciclo de vida, la cual defina las fases y acciones apropiadas.
- Crear una plantilla de índice, para crear así el flujo de datos y aplicar la política de ILM, además de la configuración de índices y asignaciones para los índices de respaldo.
- Verificar que los índices se muevan a través de las fases del ciclo de vida tal y como se esperaba.

Una política del ciclo de vida especifica las fases del ciclo de vida del índice y las acciones a realizar en cada fase. Puede tener hasta cuatro fases: *hot*, *warm*, *cold* y *delete*.

Por ejemplo, es posible definir un *timeseries_policy* que tenga dos fases:

- Una fase *hot* que establezca un *rollover* cuando el índice alcance un *max_size* de 50 gigabytes o un *max_age* de 30 días.
- Una fase *delete* que establece un *min_age*, para eliminar el índice 90 días después del *rollover*. Hay que tener en cuenta que este valor es relativo al tiempo de *rollover* y no al tiempo de creación del índice.

Cabe reseñar que también pueden realizarse acciones tales como:

- *Shrink*: Consiste en reducir la cantidad de fragmentos primarios en un índice.
- *Force Merge*: Es posible activar manualmente una fusión para reducir el número de segmentos en cada fragmento de un índice, liberando así el espacio utilizado por los documentos eliminados.
- *Freeze*: Es posible hacer que un índice sea solo de lectura, minimizando así su huella de memoria.

Es posible crear esta política con la API de políticas, tal y como se ve en la Figura 3-22. Cabe reseñar que en el cuerpo de solicitud del *rollover* también puede incluirse la condición *max_docs*.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/_ilm/policy/timeseries_policy?pretty" -H 'Content-Type: application/json' -d'
> {
>   "policy": {
>     "phases": {
>       "hot": {
>         "actions": {
>           "rollover": {
>             "max_size": "50GB",
>             "max_age": "30d"
>           }
>         }
>       },
>       "delete": {
>         "min_age": "90d",
>         "actions": {
>           "delete": {}
>         }
>       }
>     }
>   }
> }
>
> "acknowledged" : true
```

Figura 3-22 Creación de política de ciclo de vida

Para crear un flujo de datos primero hay que crear un *index_template* para especificar la política del ciclo de vida y una definición del *data_stream*.

En este caso creamos una plantilla *timeseries_template* para usarlo en futuros flujos de datos, denominada *timeseries*.

Para habilitar el ILM y administrar el flujo de datos, en la plantilla ha de indicarse el nombre de la política de ciclo de vida que se va a aplicar (*Index.lifecycle.name*).

En la Figura 3-23, se puede ver la creación de la plantilla de índice a través de la API de plantilla de índice.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/_index_template/timeseries_template?pretty" -H 'Content-Type: application/json' -d'
{
  "index_patterns": ["timeseries"],
  "data_stream": { },
  "template": {
    "settings": {
      "number_of_shards": 2,
      "number_of_replicas": 1,
      "index.lifecycle.name": "timeseries_policy"
    }
  }
},
{
  "acknowledged" : true
}
```

Figura 3-23 Creación plantilla de índice (*index template*)

Para crear un flujo de datos, hay que indexar un documento con el nombre o patrón comodín definido en *index_patterns* de la plantilla índice. Siempre que un flujo de datos, índice o alias de índice existente, no utilice el nombre definido en *index_patterns*, la solicitud de índice crea automáticamente un flujo de datos correspondiente con un único índice de respaldo. Elasticsearch indexa automáticamente los documentos de la solicitud en este índice de respaldo, el cual también actúa como índice de escritura.

Por ejemplo, como se puede ver en la Figura 3-24, la siguiente solicitud crea el flujo de datos *timeseries* y el índice de respaldo de primera generación llamado *.ds-timeseries-2021.02.13-000001*.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X POST "localhost:9200/timeseries/_doc?pretty" -H 'Content-Type: application/json' -d'
{
  "message": "logged the request",
  "@timestamp": "1591890611"
},
{
  "_index" : ".ds-timeseries-2021.02.13-000001",
  "_type" : "_doc",
  "_id" : "banrtHcBweY5z46ho2Tk",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 3,
  "_primary_term" : 6
}
```

Figura 3-24 Creación de un flujo de datos

Cuando se cumple una condición de *rollover* en la política de ciclo de vida, la acción de renovación sería la siguiente:

- Se crea el índice de respaldo de segunda generación, denominado *.ds-timeseries-2021.02.13-000002*. Debido a que es un índice de respaldo del flujo de datos de series temporales, la configuración de la plantilla de índice *timeseries_template* se aplica al nuevo índice.
- Como es el índice de última generación del flujo de datos de series temporales, el índice de respaldo recién creado *.ds-timeseries-2021.02.13.000002* se convierte en el índice de escritura del flujo de datos.

Este proceso se repite cada vez que se cumple una condición de *rollover*. Es posible buscar en todos los índices de respaldo del flujo de datos, administrados por *timeseries_policy* con el nombre del flujo de datos de *timeseries*. Las operaciones de escritura están enrutadas al actual índice de escritura y las operaciones de lectura serán manejadas por todos los índices de respaldo.

Se puede comprobar el progreso del ciclo de vida, a través de la API de ILM. Esto permite saber cosas tales como:

- En qué fase se encuentra un índice y cuando entró en esa fase.
- La situación actual y qué paso se está realizando.
- Si se ha producido algún error o si el proceso está bloqueado.

A través de la solicitud que se ve en la Figura 3-25, es posible obtener información sobre los flujos de datos *timeseries* de los índices de respaldo. La respuesta muestra que el índice de respaldo de primera generación del flujo de datos está esperando la acción de *rollover* de la fase *hot*, de tal forma que permanece en este estado hasta que se cumpla una condición de *rollover*.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/.ds-timeser
ies-*/_ilm/explain?pretty"
{
  "indices" : {
    ".ds-timeseries-2021.02.13-000001" : {
      "index" : ".ds-timeseries-2021.02.13-000001",
      "managed" : true,
      "policy" : "timeseries_policy",
      "lifecycle_date_millis" : 1613251297397,
      "age" : "4.59d",
      "phase" : "hot",
      "phase_time_millis" : 1613251299045,
      "action" : "rollover",
      "action_time_millis" : 1613251719511,
      "step" : "check-rollover-ready",
      "step_time_millis" : 1613251719511,
      "phase_execution" : {
        "policy" : "timeseries_policy",
        "phase_definition" : {
          "min_age" : "0ms",
          "actions" : {
            "rollover" : {
              "max_size" : "50gb",
              "max_age" : "30d"
            }
          }
        }
      },
      "version" : 1,
      "modified_date_in_millis" : 1613251167611
    }
  }
}
```

Figura 3-25 Comprobación del estado del ciclo de vida

3.1.5 Consultas geográficas

Elasticsearch admite dos tipos de datos geográficos: campos *geo_point*, que admiten pares latitud/longitud, y campos *geo_shape*, que admiten puntos, líneas, círculos, polígonos, etc.

Las consultas en este grupo son:

- Consulta *geo_bounding_box*: Busca documentos con puntos geográficos que caen en el rectángulo especificado.

- Consulta *geo_distance*: Busca documentos con puntos geográficos dentro de la distancia especificada desde un punto central.
- Consulta *geo_polygon*: Buscar documentos con puntos geográficos dentro del polígono especificado.
- Consulta *geo_shape*: Busca documentos con:
 - *Geo_shapes* que se cruzan, están contenidas, o no se cruzan con la forma geométrica especificada.
 - *Geo_points* que se cruzan con la forma geométrica especificada.

Para ver un ejemplo de todas las consultas geográficas que nos ofrece Elasticsearch, indexamos los documentos que se pueden ver en la Figura 3-26 y en la Figura 3-27, que utilizaremos para las dos primeras consultas geográficas (*geo_bounding_box* y *geo_distance*). En el siguiente ejemplo se están indexando tanto un campo *geo_point*, delimitado por coordenadas latitud/longitud (40.12/-71.34), como un campo *geo_shape*, en el que se indican las coordenadas de un polígono.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/my_location
s?pretty" -H 'Content-Type: application/json' -d'
> {
>   "mappings": {
>     "properties": {
>       "pin": {
>         "properties": {
>           "location": {
>             "type": "geo_point"
>           }
>         }
>       }
>     }
>   }
> }
> '
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "my_locations"
}
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/my_location
s/_doc/1?pretty" -H 'Content-Type: application/json' -d'
> {
>   "pin": {
>     "location": {
>       "lat": 40.12,
>       "lon": -71.34
>     }
>   }
> }
> '
{
  "_index" : "my_locations",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Figura 3-26 Consulta *geo_point* y *geo_shape*

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/my_geoshape
s?pretty" -H 'Content-Type: application/json' -d'
> {
>   "mappings": {
>     "properties": {
>       "pin": {
>         "properties": {
>           "location": {
>             "type": "geo_shape"
>           }
>         }
>       }
>     }
>   }
> },
> {
>   "acknowledged" : true,
>   "shards_acknowledged" : true,
>   "index" : "my_geoshapes"
> }
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/my_geoshape
s/_doc/1?pretty" -H 'Content-Type: application/json' -d'
> {
>   "pin": {
>     "location": {
>       "type": "polygon",
>       "coordinates" : [[[13.0 ,51.5], [15.0, 51.5], [15.0, 54.0], [13.0, 54.0], [13.0 ,51.5]]]
>     }
>   }
> },
> {
>   "_index" : "my_geoshapes",
>   "_type" : "_doc",
>   "_id" : "1",
>   "_version" : 1,
>   "result" : "created",
>   "_shards" : {
>     "total" : 2,
>     "successful" : 1,
>     "failed" : 0
>   },
>   "_seq_no" : 0,
>   "_primary_term" : 1
> }
}
```

Figura 3-27 Continuación consulta *geo_point* y *geo_shape*

A continuación, se detalla un ejemplo de cada una de las consultas geográficas: *geo_bounding_box*, *geo_distance*, *geo_polygon* y *geo_shape*.

3.1.5.1 Geo_bounding_box

Para encontrar los *geo_points* que se encuentran dentro de un *bounding box* hay que utilizar un filtro *geo_bounding_box*. Para definir el cuadro (*box*), se deben proporcionar valores de puntos geográficos para dos esquinas opuestas, como se puede ver en la Figura 3-28.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/my_location/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": {
        "match_all": {}
      },
      "filter": {
        "geo_bounding_box": {
          "pin.location": {
            "top_left": {
              "lat": 40.73,
              "lon": -74.1
            },
            "bottom_right": {
              "lat": 40.01,
              "lon": -71.12
            }
          }
        }
      }
    }
  }
}
```

Figura 3-28 Consulta *geo_bounding_box* y definición del *box*

Usamos el mismo filtro para hacer coincidir los valores *geo_shape* que se solapan con el *bounding box*, de tal forma que en vez de poner *my_locations*, habría que escribir *my_geoshapes*. Posteriormente, recuperamos tanto los *geo_point*, como los *geo_shape*, tal y como se ve en la Figura 3-29.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/my_location_s,my_geoshapes/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": {
        "match_all": {}
      },
      "filter": {
        "geo_bounding_box": {
          "pin.location": {
            "top_left": {
              "lat": 40.73,
              "lon": -74.1
            },
            "bottom_right": {
              "lat": 40.01,
              "lon": -71.12
            }
          }
        }
      }
    }
  }
}
```

Figura 3-29 Consulta para recuperar *geo_point* y *geo_shape*

3.1.5.2 Geo_distance

A partir del documento indexado que se puede ver en la Figura 3-26, hacemos la consulta *geo_distance*, para recuperar los valores *geo_point* dentro de una distancia especificada por otro *geo_point*, como puede verse en la Figura 3-30.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/my_location
s/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": {
>     "bool": {
>       "must": {
>         "match_all": {}
>       },
>       "filter": {
>         "geo_distance": {
>           "distance": "200km",
>           "pin.location": {
>             "lat": 40,
>             "lon": -70
>           }
>         }
>       }
>     }
>   }
> }
```

Figura 3-30 Consulta *geo_distance*

Aplicamos el mismo filtro para recuperar los *geo_shape* que se solapan con el área circular establecida por las coordenadas lat/lon y la distancia indicada. Posteriormente, recuperamos tanto los *geo_point*, como los *geo_shape*, como se observa en la Figura 3-31.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/my_location
s,my_geoshapes/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": {
>     "bool": {
>       "must": {
>         "match_all": {}
>       },
>       "filter": {
>         "geo_distance": {
>           "distance": "200km",
>           "pin.location": {
>             "lat": 40,
>             "lon": -70
>           }
>         }
>       }
>     }
>   }
> }
```

Figura 3-31 Consulta para recuperar *geo_point* y *geo_shape*

3.1.5.3 Geo_polygon

Esta consulta devuelve todos los datos que se encuentran dentro de un polígono definido por una serie de puntos. En la Figura 3-32, puede verse un ejemplo de este tipo de consulta, donde Elasticsearch nos devolvería todos los datos que estuvieran dentro del triángulo formado por las coordenadas que se ven en la Figura 3-32.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": {
        "match_all": {}
      },
      "filter": {
        "geo_polygon": {
          "person.location": {
            "points": [
              { "lat": 40, "lon": -70 },
              { "lat": 30, "lon": -80 },
              { "lat": 20, "lon": -90 }
            ]
          }
        }
      }
    }
  }
}
```

Figura 3-32 Consulta *geo_polygon*

3.1.5.4 Geo_shape

Permite filtrar documentos indexados usando *geo_shape* y *geo_point*. Además, requiere *geo_shape mapping* o *geo_point mapping*. Cabe recordar que *mapping* es el proceso de definir cómo se almacena e indexa un documento y los campos que contiene.

La consulta *geo_shape* utiliza la misma representación de cuadrícula que el *geo_shape mapping*, para encontrar documentos que tengan una forma que se solape con la forma de la consulta.

Se admiten dos maneras de definir la forma de la consulta, ya sea proporcionando una definición de forma completa o haciendo referencia al nombre de una forma pre-indexada en otro índice.

Ambos formatos pueden verse a continuación en las figuras, siendo el primero de los formatos *inline shape definition*, similar al tipo *geo_shape*. La consulta *geo_shape* usa *GeoJSON* para representar formas. En la Figura 3-33, se puede observar un índice con ubicaciones como campos *geo_shape*, siendo similar para *geo_points*.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/example?pretty" -H 'Content-Type: application/json' -d'
{
  "mappings": {
    "properties": {
      "location": {
        "type": "geo_shape"
      }
    }
  }
}
'
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "example"
}
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X POST "localhost:9200/example/_doc?refresh&pretty" -H 'Content-Type: application/json' -d'
{
  "name": "Wind & Wetter, Berlin, Germany",
  "location": {
    "type": "point",
    "coordinates": [ 13.400544, 52.530286 ]
  }
}
'
{
  "_index" : "example",
  "_type" : "_doc",
  "_id" : "bqkVtXcBweY5z46hG2RT",
  "_version" : 1,
  "result" : "created",
  "forced_refresh" : true,
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Figura 3-33 Definición *mapping geo_shape* e introducción de datos

La consulta mostrada en la Figura 3-34, encontrará el punto usando la extensión *GeoJSON* de Elasticsearch *envelope*.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/example/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": {
>     "bool": {
>       "must": {
>         "match_all": {}
>       },
>       "filter": {
>         "geo_shape": {
>           "location": {
>             "shape": {
>               "type": "envelope",
>               "coordinates": [ [ 13.0, 53.0 ], [ 14.0, 52.0 ] ]
>             },
>             "relation": "within"
>           }
>         }
>       }
>     }
>   }
> }
>
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "example",
        "_type" : "_doc",
        "_id" : "bqkVtXcBweY5z46hG2RT",
        "_score" : 1.0,
        "_source" : {
          "name" : "Wind & Wetter, Berlin, Germany",
          "location" : {
            "type" : "point",
            "coordinates" : [
              13.400544,
              52.530286
            ]
          }
        }
      }
    ]
  }
}
```

Figura 3-34 Consulta geo_shape

El otro formato es *pre-indexed shape* (forma pre-indexada), la consulta también admite el uso de una forma que ya se ha indexado en otro índice, resultando de gran utilidad para cuando se tiene una lista predefinida de formas y se desea hacer referencia a la lista con un nombre lógico (por ejemplo, España), en lugar de tener que proporcionar coordenadas cada vez. En este caso habría que incluir los siguientes parametros:

- ID: El ID del documento que contiene la forma pre-indexada.
- *Index* (Índice): El nombre del índice donde se encuentra la forma pre-indexada.
- *Path* (Ruta): La ruta que contiene la forma pre-indexada.
- *Routing* (Enrutamiento): El enrutamiento del documento *shape*, si es necesario.

En la Figura 3-35 y Figura 3-36 , se puede ver un ejemplo usando el filtro con un *pre-indexed shape*.

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/shapes?pretty" -H 'Content-Type: application/json' -d'
{
  "mappings": {
    "properties": {
      "location": {
        "type": "geo_shape"
      }
    }
  }
}
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "shapes"
}
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X PUT "localhost:9200/shapes/_doc/deu?pretty" -H 'Content-Type: application/json' -d'
{
  "location": {
    "type": "envelope",
    "coordinates" : [[13.0, 53.0], [14.0, 52.0]]
  }
}
{
  "_index" : "shapes",
  "_type" : "_doc",
  "_id" : "deu",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Figura 3-35 Consulta *pre-indexed shape*

```
javier@javier-VirtualBox:~/Documentos/elasticsearch-7.11.0$ curl -X GET "localhost:9200/example/_search?pretty" -H 'Content-Type: application/json' -d'
> {
>   "query": {
>     "bool": {
>       "filter": {
>         "geo_shape": {
>           "location": {
>             "indexed_shape": {
>               "index": "shapes",
>               "id": "deu",
>               "path": "location"
>             }
>           }
>         }
>       }
>     }
>   },
>   "took" : 192,
>   "timed_out" : false,
>   "_shards" : {
>     "total" : 1,
>     "total" : 1,
>     "successful" : 1,
>     "skipped" : 0,
>     "failed" : 0
>   },
>   "hits" : {
>     "total" : {
>       "value" : 1,
>       "relation" : "eq"
>     },
>     "max_score" : 0.0,
>     "hits" : [
>       {
>         "_index" : "example",
>         "_type" : "_doc",
>         "_id" : "bqkVtXcBweY5z46hG2RT",
>         "_score" : 0.0,
>         "_source" : {
>           "name" : "Wind & Wetter, Berlin, Germany",
>           "location" : {
>             "type" : "point",
>             "coordinates" : [
>               13.400544,
>               52.530286
>             ]
>           }
>         }
>       }
>     ]
>   }
> }
```

Figura 3-36 Consulta recuperación de *geo_shape*

3.1.5.4.1 Relaciones espaciales

El parámetro *geo_shape strategy mapping* (mapeo de estrategia *geo_shape*) determina qué operador de relación espacial puede ser usado en la búsqueda.

Los operadores de relaciones espaciales disponibles al buscar un campo de tipo *geo_shape* son los siguientes:

- *Intersects*: Devuelve todos los documentos cuyo campo *geo_shape* interseca la geometría de la consulta.
- *Within*: Devuelve todos los documentos cuyo campo *geo_shape* esté dentro de la consulta.

- *Contains*: Devuelve todos los documentos cuyo campo *geo_shape* contiene la geometría de la consulta.
- *Disjoint*: Devuelve todos los documentos cuyo campo *geo_shape* no tiene nada en común con la geometría de la consulta.

En cambio, en la búsqueda de campos tipo *geo_point*, solo hay un operador de relación espacial permitido:

- *Intersects*: Devuelve todos los documentos cuyo campo *geo_point* interseca la geometría de la consulta.

4 PRUEBA

El objetivo de este capítulo es presentar los resultados de una prueba de Elasticsearch operando sobre una captura de datos obtenidos a partir de un flujo real de tramas AIS proporcionado por el COVAM. Estos datos se indexaron en la herramienta, realizándose a continuación una serie de consultas de las que se mide el tiempo de respuesta.

4.1.1 Fichero de tramas AIS

El fichero utilizado para realizar las pruebas en este proyecto contiene 5 millones de entradas, representando tramas AIS de clase A obtenidas en un rango de aproximadamente 29 horas entre el 14 y el 15 de febrero de 2021. Este fichero está en formato CSV (*Comma-Separated Values*), de tal forma que los diferentes campos están divididos por comas y hay una trama por línea.

Podemos describir los campos de las líneas del fichero a través del siguiente ejemplo:

1613278877000, 367371820, 32, 33, 33, 32.20492, -80.27646, 8944d08a5dbffff, NOMAC, NOMIC, 559, 91.

- El primer campo (*1613278877000*), es un sello temporal. Mide los milisegundos desde el 1 de enero de 1970.
- El segundo campo (*367371820*), es el MMSI del barco. Lo consideraremos como el identificador de los barcos en las diferentes consultas.
- El tercer campo (*32*), indica el tipo de embarcación, el número 0 equivaldría a “no disponible”. [113]
- El cuarto campo (*33*), es la eslora declarada del barco, el número 0 equivaldría a “no disponible”. La eslora es la suma de A+B en la referencia. [113]
- El quinto campo (*32.20492, -80.27646*), indica la latitud y longitud del barco en el momento marcado por el sello temporal.
- El sexto campo, (*8944d08a5dbffff*), es un identificador de la celda en la que se encuentra geo-posicionado el barco según el esquema Uber H3. [114]
- El séptimo campo indica macro-zonas. Muestran si el barco se encuentra en *PENINSULA* o *CANARIAS*; o de lo contrario, si la zona no es de interés, *NOMAC*. Estas zonas son circulares, *PENINSULA* tiene centro en GPS (40.36, -3.71) y un radio de 435 nm, mientras que *CANARIAS* tiene centro en GPS (28.2810, -15.9575) y radio 180 nm.
- El octavo campo indica micro-zonas. Muestran si el barco se encuentra en *VALENCIA*, *ALGECIRAS*, *BARCELONA*, *LAS_PALMAS*, *BILBAO*, *TENERIFE*, *CARTAGENA*, *ALMERIA*, *GIJON*, *VIGO*; o de lo contrario si la zona no es de interés, *NOMIC*. Para que aparezca reflejado que el barco está cerca del puerto, se utiliza un círculo con centro en las coordenadas GPS de la ciudad correspondiente (Según Wikipedia [115]) y radio de 3 nm.

- El noveno campo (559), indica el COG (*course over ground*) declarado por el barco al transmitir en AIS. El valor 3600 equivaldría a “no disponible”. [113]
- Por último, el décimo campo (91), indica el SOG (*speed over ground*) declarado por el barco al transmitir en AIS. El valor 1023 equivaldría a “no disponible”. [113]

Es necesario tener Python instalado, ya que utilizaremos un *script* en Python, el cual procesa un fichero CSV e indexa los documentos. Este script permite operar línea a línea, de tal forma que se añaden los documentos uno a uno, simulando una situación en la que las tramas AIS se irán indexando una a una a medida que llegan por el flujo. Este script trabaja con un índice de nombre “aisdata”.

Para poder ejecutarlo, son necesarias una serie de librerías, las cuales se pueden instalar utilizando los siguientes comandos:

```
-$ sudo pip install Elasticsearch (cliente Python de Elasticsearch)
```

```
-$ sudo pip install pytest-timeit (librería para medición de tiempos)
```

```
-$ sudo pip install python-csv (librería para procesado de ficheros CSV)
```

Tras instalar las librerías, ejecutamos el script de la siguiente manera:

```
-$ python indexDocs.py fichero-pruebas.csv>salida
```

El script lee todos los datos del fichero y va escribiendo mensajes, que se volcarán en el fichero de texto “salida”. Al abrir el fichero se puede ver que tiene una serie de líneas que contienen el tiempo que se tardó en indexar un documento, y la última línea contiene el tiempo total (en segundos) y el número de documentos procesados. Cabe destacar que el tiempo total en indexar las 5 millones de trazas AIS ha sido 2 horas y 16 minutos.

A continuación, se describe el código del *script indexDocs.py*:

```
# Para poder acceder al fichero CSV y a los parámetros específicos del sistema.
```

```
import sys
import csv
```

```
# Se importan las librerías que se van a utilizar, las cuales fueron instaladas anteriormente.
```

```
from timeit import default_timer as timer
from elasticsearch import Elasticsearch
```

```
# Indicamos el nombre del índice que queremos utilizar.
```

```
_index_name="aisdata"
```

```
#Datos de conexión de Elasticsearch.
```

```
_host="localhost"
_port=9200
```

```
# Inicializamos la conexión con Elasticsearch, indicando host y puerto del servidor.
```

```
elastic = Elasticsearch([{'host': _host, 'port': _port}])
```

```
# Primero se empieza borrando el índice, por si ya existiese alguna prueba anterior. Se ignoran los errores.
```

```
elastic.indices.delete(index=_index_name, ignore=404)
```

Configuramos el índice que vamos a crear.

```
data = {
    'settings': {
        'number_of_shards': 2,
        'number_of_replicas': 1
    },
    'mappings': {
        'properties': {
            '@timestamp': {'type': 'date'},
```

El timestamp se guarda también como wildcard para consultas de prefijo.

```
            'timecard': {'type': 'wildcard'},
            'mmsi': {'type': 'long'},
            'type': {'type': 'integer'},
            'length': {'type': 'integer'},
            'position': {'type': 'geo_point'},
            'cell': {'type': 'wildcard'},
            'macro': {'type': 'keyword'},
            'micro': {'type': 'keyword'},
            'cog': {'type': 'integer'},
            'sog': {'type': 'integer'}
        }
    }
}
```

Creamos el índice.

```
elastic.indices.create(index=_index_name, body=data)
```

Contamos el número de documentos procesados y el tiempo total del procesamiento.

```
counter = 0
start_total = timer()
```

Procesamos el fichero de datos cargando cada línea como un documento.

```
with open(sys.argv[1]) as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        doc = {'@timestamp': int(row[0]), 'timecard': str(row[0]),
            'mmsi': int(row[1]), 'type': int(row[2]), 'length':
            int(row[3]), 'position': {'lat': float(row[4]), 'lon':
            float(row[5])}, 'cell': row[6], 'macro': row[7], 'micro':
            row[8], 'cog': int(row[9]), 'sog': int(row[10])}

        start = timer()
        response = elastic.index(index=_index_name, body=doc)
        end = timer()
        counter = counter+1
        print(end-start, response['result'])
end_total = timer()
```

```
# Finalmente, imprimimos el tiempo total y el número de documentos procesados.
print(end_total-start_total, counter)
```

4.1.2 Tipos de consultas

Las pruebas se han llevado a cabo en un servidor ya que los tiempos de ejecución en la máquina virtual son más grandes y menos realistas debido al impacto de la virtualización. Los datos del servidor son los siguientes:

- Sistema operativo: Ubuntu 5.4.0-42 generic x86_64 GNU/Linux.
- Memoria: 370-AEQH RDIMM de 32 GB, 2933 MT/s (x8).
- CPU: Intel(R) Xeon(R) Bronze 3204 CPU @ 1.90Ghz (x2).
- Disco: 400-AXSK SSD SATA Read-Intensive 3,84 TB 6Gb/s (x6).

Para acceder al servidor se hace a través de SSH, introduciendo un usuario y contraseña, tal y como se ve en la Figura 4-1:

```
-$ssh -p1024 tfg@193.146.212.7
```

```
[MacBook-Air-de-Javier-2:~ javiervila$ ssh -p1024 tfg@193.146.212.7
[tfg@193.146.212.7's password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-42-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Thu Mar  4 17:20:55 UTC 2021

System load:  0.11           Processes:            278
Usage of /:   0.5% of 17.39TB Users logged in:        1
Memory usage: 15%           IPv4 address for eno3: 192.168.16.20
Swap usage:   0%            IPv4 address for tun0: 10.110.110.3
Temperature: 36.0 C

 * Introducing self-healing high availability clusters in MicroK8s.
   Simple, hardened, Kubernetes for production, from RaspberryPi to DC.

   https://microk8s.io/high-availability

82 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

*** System restart required ***
Last login: Thu Mar  4 17:05:40 2021 from 193.152.116.197
tfg@cavam:~$ █
```

Figura 4-1 Acceso al servidor

Para hacer las pruebas se han utilizado una serie de *scripts* de Python para cada uno de los tipos de consulta que se van a llevar a cabo. Los *scripts* imprimen una línea de texto a la salida, que tiene el siguiente aspecto:

```
PORT, 0.059497, 4405, VIGO
```

Donde, el primer campo representa el tipo de consulta: “ALL” (todo), “PORT” (microzonas), “LENGTH” (eslora), “POS” (posiciones), “CELL” (celdas H3), “TIME” (rango de tiempo medido en

minutos desde el sello temporal inicial), “MMSI”, “NEAR” (proximidad por celda H3), “ZONE” (geozona), “NEARGEO” (proximidad por coordenadas GPS)

El segundo campo nos indica el tiempo que tarda en ejecutarse la consulta, el cual utilizaremos para valorar los resultados y sacar unas estadísticas, a través de la media, desviación típica, mediana, valor máximo y valor mínimo.

El tercer campo nos indica el número de resultados obtenidos, cabe reseñar que Elasticsearch devuelve por defecto (aunque es posible configurarlo) un número de resultados máximo igual a 10000, aunque la búsqueda la realice sobre los 5 millones de entradas. Y, por último, el cuarto campo indica los parámetros empleados en la consulta, en el ejemplo de arriba se realizó una consulta de los barcos que han estado en la micro-zona del puerto de Vigo.

El código de los distintos *scripts* es muy parecido a excepción de la consulta, es por ello que solo se va a explicar el código de uno de ellos.

```
# Para poder acceder a los parámetros específicos del sistema,
```

```
import sys
```

```
# Importamos las librerías necesarias para la medición de tiempos y comunicación con Elasticsearch.
```

```
from timeit import default_timer as timer
from elasticsearch import Elasticsearch
```

```
# Indicamos el nombre del índice que queremos utilizar,
```

```
_index_name="aisdata"
```

```
# Indicamos los datos de conexión de Elasticsearch,
```

```
_host="localhost"
_port=9200
```

```
# Especificamos el número de resultados,
```

```
_size=100
```

```
# Inicializamos la conexión con Elasticsearch indicando host y puerto del servidor,
```

```
elastic = Elasticsearch([{'host': _host, 'port': _port}])
```

```
start = timer()
results = elastic.search(index=_index_name, body={<Consulta>})
total_time = timer() - start
```

```
print("Tipo_consulta,%f,%d,%s"%(total_time, results['hits']['total']
['value'], sys.argv[1]))
```

```
# <Consulta> es lo que se reemplaza por la consulta que se desea realizar.
```

`sys.argv[1]`, indica el parámetro que introducimos en la consulta, si hay mas de uno se imprimirán a continuación

Los *scripts* utilizados son los siguientes, los cuales simplemente hay que ejecutarlos utilizando los siguientes comandos y añadiendo el parámetro o parámetros de búsqueda que proceda:

- “ALL”: “Python searchAll.py (Sin parámetro)”
- “PORT”: Python searchPort.py (Parámetro: microzona)”
- “LENGTH”: Python searchLength.py (Parametro: eslora)”
- “POS”: “Python searchPos.py (Parámetro: MMSI)”
- “CELL”: “Python searchCell.py (Parámetro: Prefijo Celda H3)”
- “TIME”: “Python searchTime.py (Parámetros: Tiempo en minutos y MMSI)”
- “MMSI”: “Python searchMMSI.py (Parámetro: MMSI)”
- “NEAR”: “Python searchNear.py (Parámetro: prefijo del sello temporal)”
- “ZONE”: “Python searchZone.py (Sin parámetro)”
- “NEARGEO”: “Python searchNearGeo.py (Parámetros: MMSI, Sello temporal y distancia en km)”

4.1.3 Análisis de las consultas

Es este apartado se detallan cada uno de los tipos de consultas presentados en el apartado 4.1.2. Se ha probado cada una 30 veces (con diferentes parámetros en aquellas consultas que lo permiten) para poder sacar unas estadísticas del tiempo de ejecución obtenido, indicando la media, desviación típica, mediana, valor máximo y valor mínimo, además de un diagrama de barras con los resultados obtenidos. Los resultados obtenidos para cada una de las consultas realizadas se pueden ver en el Anexo I: Resultados obtenidos

4.1.3.1 Consulta ALL

Al ejecutar el *script* “searchAll.py” obtenemos el tiempo de ejecución en buscar todas las tramas AIS que se encuentran en el fichero de 5 millones de entradas.

La consulta utilizada en el *script* es la siguiente:

```
{"query": {"match_all": {}},
  "size": _size}
```

Al ejecutar el script como:

```
python searchAll.py
```

Nos da una respuesta como la que se indica a continuación:

```
ALL,0.022815,10000
```

En la siguiente tabla se muestran una serie de estadísticas obtenidas del tiempo de ejecución (medido en segundos) tras ejecutar el script 30 veces.

MEDIA	0,01249103
DESVIACIÓN TÍPICA	0,00259358
MEDIANA	0,0124715
VALOR MÁXIMO	0,023125
VALOR MÍNIMO	0,008464

Tabla 6 Estadísticas consultas ALL

En la Figura 4-2, se puede observar como han ido variando los tiempos de ejecución. Normalmente la primera búsqueda requiere un tiempo de ejecución mayor, y en este caso el resto se mantienen aproximadamente estables.

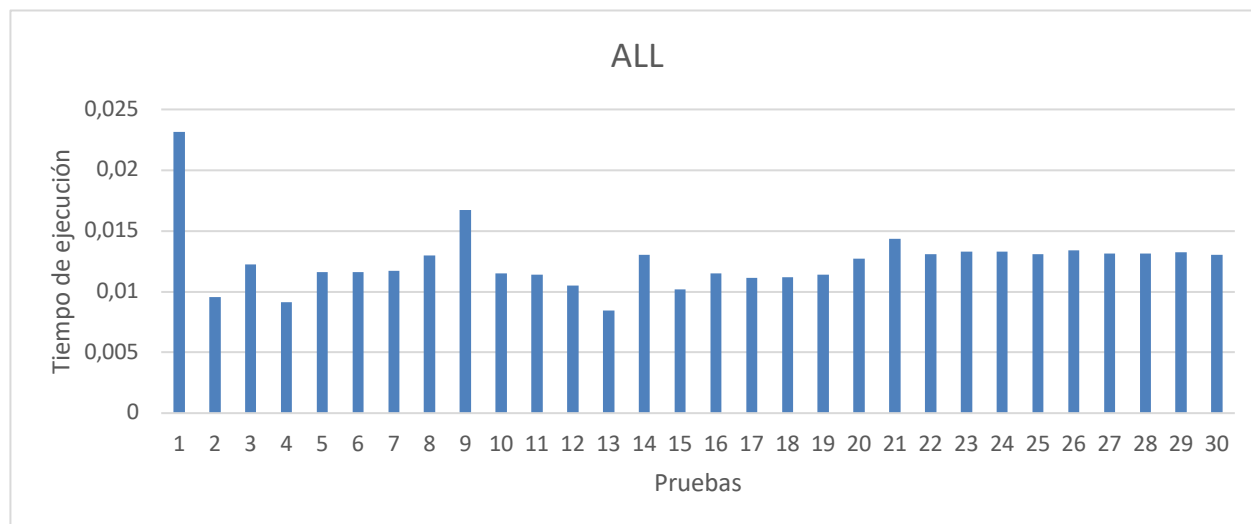


Figura 4-2 Diagrama de barras consultas ALL

4.1.3.2 Consulta PORT

Al ejecutar el *script* “searchPort.py” obtenemos el tiempo de ejecución en buscar todos los barcos que han estado en el puerto (micro-zona) que nosotros especifiquemos en la consulta. Las opciones posibles son Vigo, Barcelona, Cartagena, Almería, Las Palmas, Algeciras, Bilbao, Gijón, Tenerife y Valencia.

La consulta utilizada en el *script* es la siguiente:

```
{"query": {"match": {"micro": sys.argv[1]}},
"size": _size}
```

Al ejecutar el *script* como:

```
python searchPort.py VIGO
```

Nos da una respuesta como la que se indica a continuación:

```
PORT,0.025447,4405,VIGO
```

Lo cual quiere decir que ha habido 4405 tramas AIS dentro del radio de 3 nm alrededor de Vigo.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el *script* 30 veces. Se han introducido los 10 puertos tres veces cada uno de manera aleatoria en las consultas.

MEDIA	0,02510865
DESVIACIÓN TÍPICA	0,00665991
MEDIANA	0,0241095
VALOR MÁXIMO	0,059497
VALOR MÍNIMO	0,021369

Tabla 7 Estadísticas consultas PORT

En la Figura 4-3, puede verse como la primera consulta requiere un mayor tiempo de ejecución y el resto se mantienen estables alrededor de 0,025 segundos.

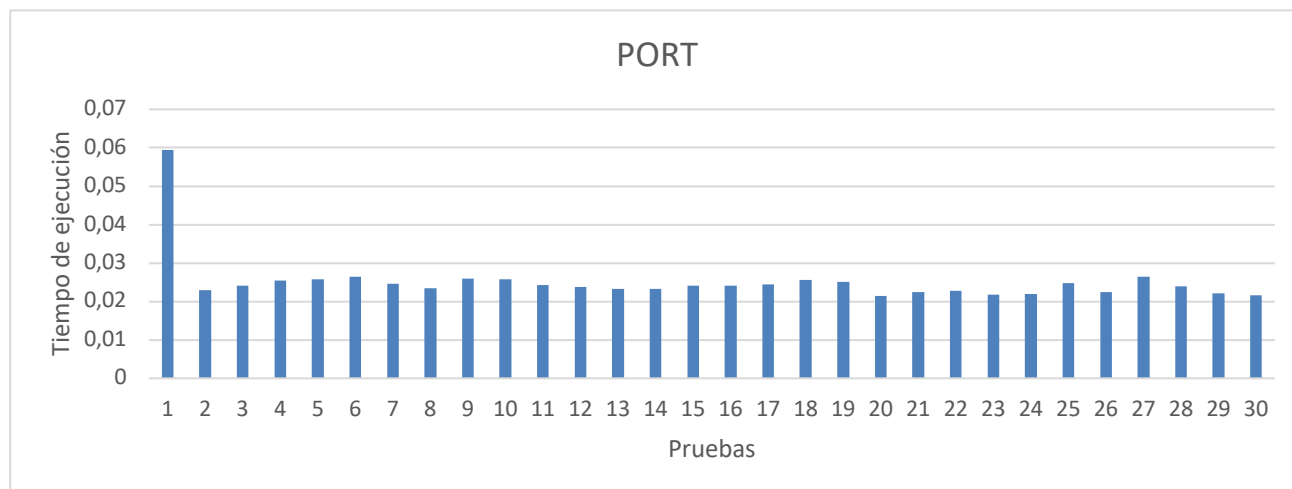


Figura 4-3 Diagrama de barras consultas PORT

4.1.3.3 Consulta LENGTH

Al ejecutar el *script* “searchLength.py” obtenemos el tiempo de ejecución en buscar los MMSI de los barcos que tienen la eslora mayor o igual a la especificada en la consulta.

La consulta utilizada en el *script* es la siguiente:

```
{"size":0, "query": {"range": {"length": {"gte": int(sys.argv[1]) }}},
  "aggs": {"mmsis": {"terms": {"field": "mmsi",
  "size": _size}}}})
```

Al ejecutar el script como:

```
python searchLength.py 30
```

Nos da una respuesta como la que se indica a continuación:

```
LENGTH,0.011001,10000,30
```

Este resultado indica que hay 10000 barcos o más que tienen una eslora de 30 metros o mayor.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el script 30 veces. Se han introducido esloras de 10 en 10 metros desde 30 hasta 320 metros.

MEDIA	0,10385057
DESVIACIÓN TÍPICA	0,06508903
MEDIANA	0,100046
VALOR MÁXIMO	0,254149
VALOR MÍNIMO	0,025195

Tabla 8 Estadísticas consultas LENGTH

En la Figura 4-4 se puede ver como va disminuyendo progresivamente el tiempo de ejecución conforme va aumentando la eslora. Probablemente esto es debido a que conforme aumenta la eslora disminuye el número de barcos en el conjunto de resultados y con ello el tiempo de ejecución. Además, al usar “aggs” y haber menos resultados se tarda menos en agregarlos, lo que puede explicar que los tiempos de respuesta sean cada vez menores.

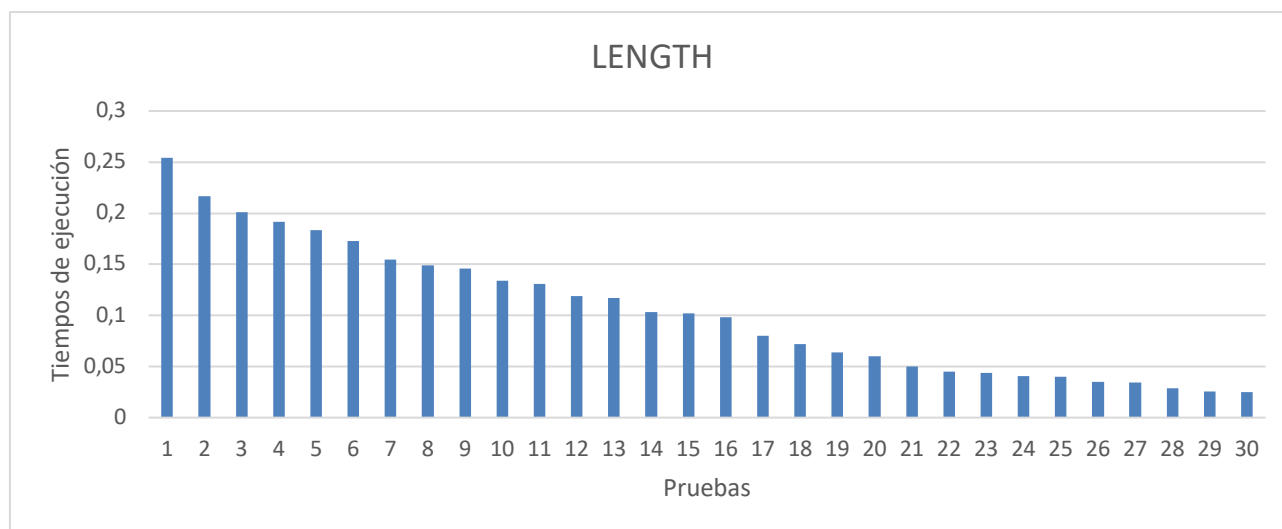


Figura 4-4 Diagrama de barras consultas LENGTH

4.1.3.4 Consulta POS

Al ejecutar el *script* “searchPos.py” obtenemos el tiempo de ejecución en buscar las últimas posiciones de un barco dado su MMSI.

La consulta utilizada en el *script* es la siguiente:

```
{"query": {"match": {"mmsi": int(sys.argv[1]) }},  
"sort": [{"@timestamp": {"order": "desc"}}],  
"fields": ["position"],  
"size": _size}
```

Al ejecutar el *script* como:

```
python searchPos.py 210401000
```

Nos da una respuesta como la que se indica a continuación:

```
POSITION,0.024950,168,210401000
```

Este resultado indica que el barco con MMSI 210401000 ha transmitido 168 veces en AIS en el tiempo recogido en el fichero de 5 millones de trazas.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el *script* 30 veces. Se han introducido MMSI de diferentes barcos de manera aleatoria.

MEDIA	0,02337813
DESVIACIÓN TÍPICA	0,00455619
MEDIANA	0,024719
VALOR MÁXIMO	0,031733
VALOR MÍNIMO	0,012301

Tabla 9 Estadísticas consultas POS

En la Figura 4-5 se puede observar como los resultados van variando en función del número de tramas AIS que haya de un barco.

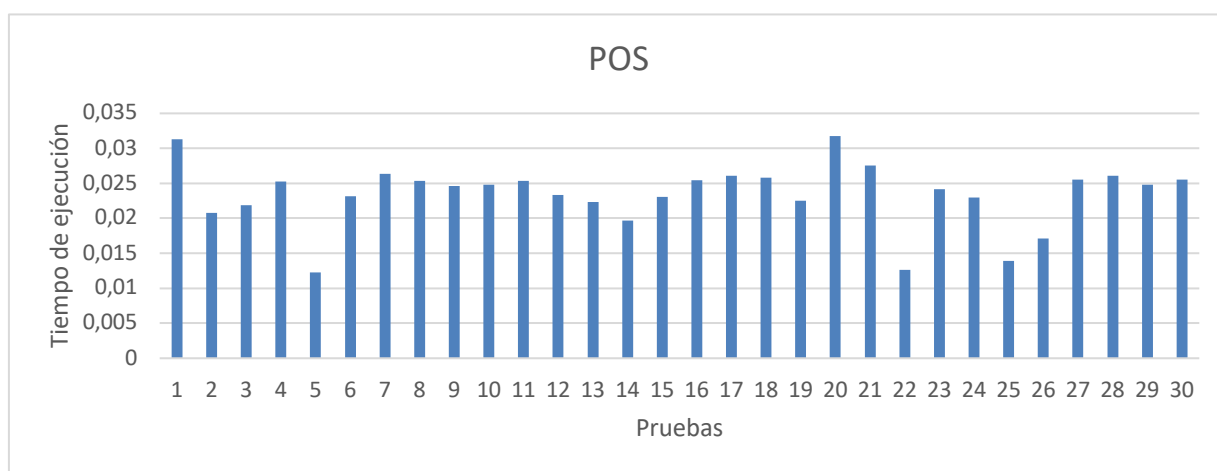


Figura 4-5 Diagrama de barras consultas POS

4.1.3.5 Consulta CELL

Al ejecutar el *script* “searchCell.py” obtenemos el tiempo de ejecución en buscar los MMSI de aquellos barcos que se encuentren dentro de una o varias celdas H3 cuyo identificador empieza con un prefijo dado.

La consulta utilizada en el *script* es la siguiente:

```
{"size":0, "query": {"prefix": {"cell": sys.argv[1] }}, "aggs": {"mmsis": {"terms": {"field": "mmsi", "size": _size}}}}
```

Al ejecutar el *script* como:

```
python searchCell.py 89195a
```

Nos da una respuesta como la que se indica a continuación:

```
CELL,0.007591,4186,89195a
```

El resultado indica que ha habido 4186 barcos dentro de la celda H3 con el prefijo 89195a, lo que incluiría por ejemplo a celdas con identificadores como: 89195a4a50fffff o 89195a1451bffff

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el *script* 30 veces. Se han introducido prefijos, de diferentes tamaños, de diferentes celdas H3 de manera aleatoria.

MEDIA	0,04977517
DESVIACIÓN TÍPICA	0,06039915
MEDIANA	0,0242055
VALOR MÁXIMO	0,216762
VALOR MÍNIMO	0,007212

Tabla 10 Estadísticas consultas CELL

En la Figura 4-6, se puede observar como varía el tiempo de ejecución en función del número de resultados, ya que el tiempo de agregación depende mucho de el número de MMSI que encuentre en las celdas especificadas por el prefijo. Cuanto más corto sea el prefijo mayor será el numero de celdas involucradas y con ello el número de resultados y el tiempo de procesamiento.

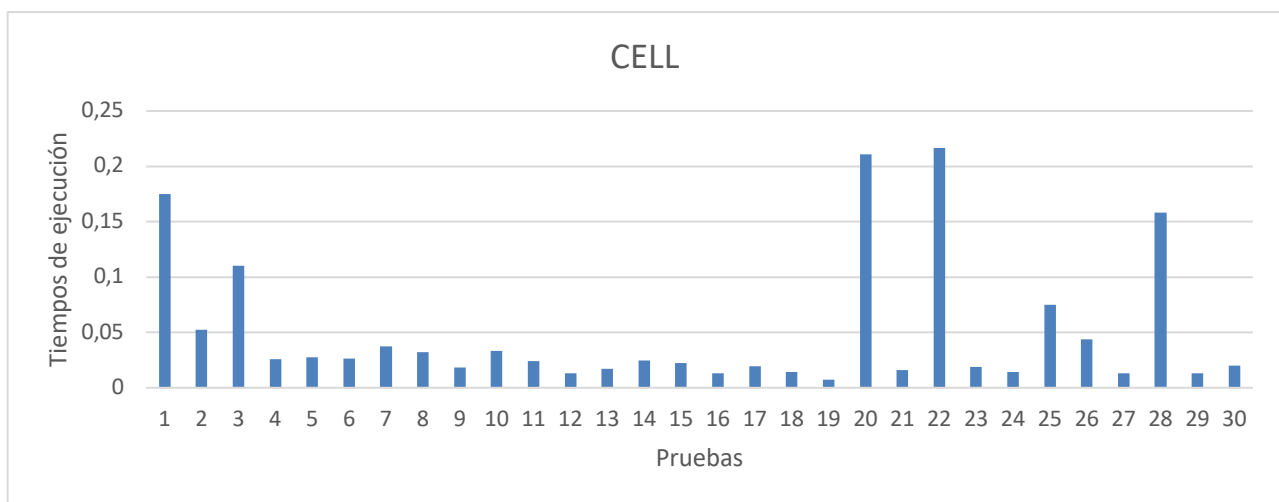


Figura 4-6 Diagrama de barras consultas CELL

4.1.3.6 Consulta TIME

Al ejecutar el *script* “searchTime.py” obtenemos el tiempo de ejecución en buscar cuantas veces ha transmitido tramas AIS un barco a lo largo de una ventana temporal de duración dada como parámetro (en minutos), tomando como inicio el sello temporal de la primera trama del fichero.

La consulta utilizada en el *script* es la siguiente:

```
{"query": {"bool": {"must": [ {"term": {"mmsi": _mmsi}},
{"range": {"@timestamp": {"gte": _inicio, "lte": _fin}}}]}}}
```

Al ejecutar el *script* como:

```
python searchTime.py 232004406 450
```

Nos da una respuesta como la que se indica a continuación:

```
TIME,0.009858,50,232004406,450
```

El resultado indica que hay 50 tramas AIS de un barco dado su MMSI, 450 minutos después del sello temporal inicial.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el *script* 30 veces. Se han introducido MMSI de manera aleatoria y tiempos crecientes desde el sello temporal inicial, desde 60 hasta 1400 minutos.

MEDIA	0,01039857
DESVIACIÓN TÍPICA	0,00215862
MEDIANA	0,0100065
VALOR MÁXIMO	0,021083
VALOR MÍNIMO	0,008227

Tabla 11 Estadísticas consultas TIME

En la Figura 4-7 se puede observar como el tiempo de ejecución de la primera consulta es mayor, y el resto se mantienen estables en torno a los 0,1 segundos.

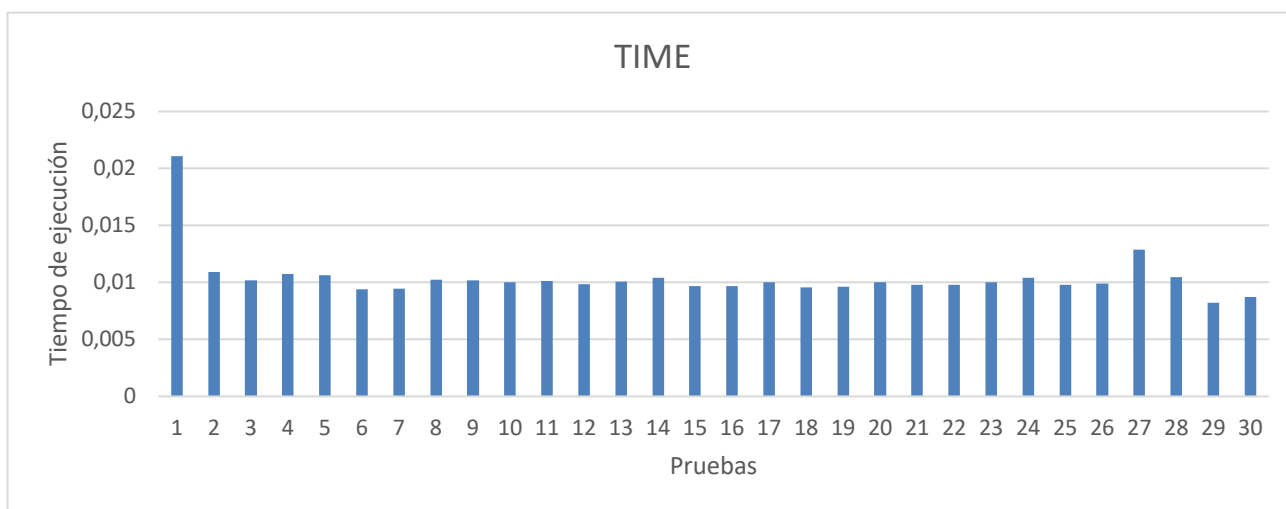


Figura 4-7 Diagrama de barras consultas TIME

4.1.3.7 Consulta MMSI

Al ejecutar el *script* “searchMMSI.py” obtenemos el tiempo de ejecución en buscar las tramas AIS de un barco dado su MMSI.

La consulta utilizada en el *script* es la siguiente:

```
{"query": {"match": {"mmsi": sys.argv[1]}},
"size": _size}
```

Al ejecutar el *script* como:

```
python searchMMSI.py 316005721
```

Nos da una respuesta como la que se indica a continuación:

```
MMSI,0.024600,163,316005721
```

El resultado obtenido indica que hay 163 transmisiones AIS del barco con MMSI 36732458 dentro del fichero de 5 millones de trazas.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el *script* 30 veces. Se han introducido MMSI de manera aleatoria.

MEDIA	0,0191374
DESVIACIÓN TÍPICA	0,00437535
MEDIANA	0,0197235
VALOR MÁXIMO	0,02582
VALOR MÍNIMO	0,008666

Tabla 12 Estadísticas consultas MMSI

En la Figura 4-8 se puede observar como varían los tiempos de ejecución en función del número de tramas AIS que haya por cada MMSI.

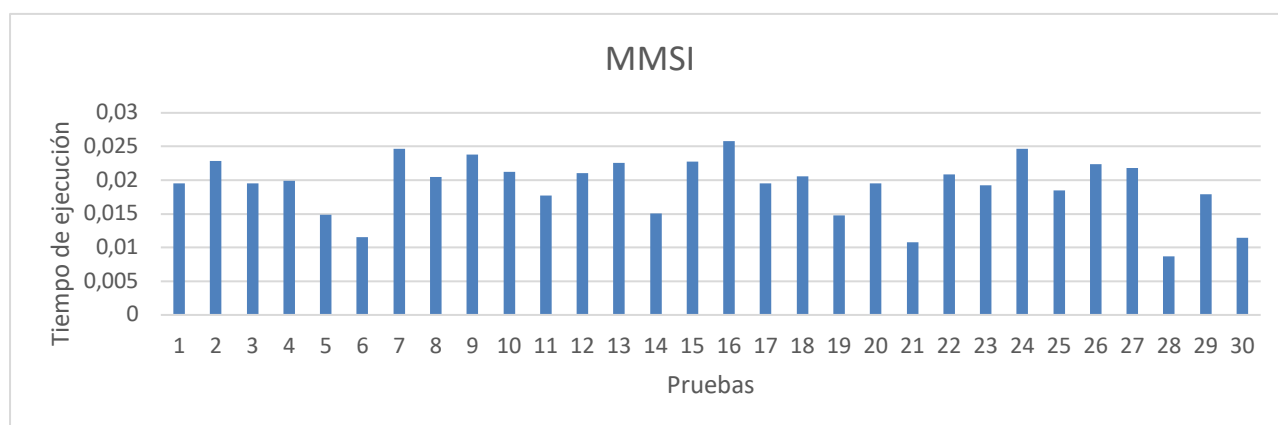


Figura 4-8 Diagrama de barras consultas MMSI

4.1.3.8 Consulta NEAR

Al ejecutar el *script* "searchNear.py" obtenemos el tiempo de ejecución en buscar MMSI de barcos que estén cerca el uno del otro en el Estrecho de Gibraltar, fijando en la consulta un prefijo que se refiere a celdas H3 ubicadas en el Estrecho y dado como parámetro un prefijo que especifica una ventana temporal.

La consulta utilizada en el *script* es la siguiente:

```
{"size":0, "query": {"bool": {"must": [ {"prefix": {"cell": "89391aaa0"}}, {"prefix": {"timecard": sys.argv[1]}}] }}, "aggs": {"mmsis": {"terms": {"field": "mmsi", "size": _size}}}}
```

Al ejecutar el *script* como:

```
python searchNear.py 1613278
```

Nos da una respuesta como la que se indica a continuación:

```
NEAR,0.006539,0,1613278
```

El resultado indica que no hay tramas AIS que se encuentren en las celdas en la ventana marcada por el prefijo del sello temporal introducido, en este caso 1613278. Cabe reseñar que la ventana temporal en este caso es de 1000 segundos, es decir, desde el instante temporal 1613278000000 hasta 1613278999999.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el script 30 veces. Se han introducido prefijos de sellos temporales de diferentes tamaños de manera aleatoria.

MEDIA	0,0117888
DESVIACIÓN TÍPICA	0,00254309
MEDIANA	0,0111025
VALOR MÁXIMO	0,023663
VALOR MÍNIMO	0,010468

Tabla 13 Estadísticas consultas NEAR

En la Figura 4-9, se puede observar como el tiempo de ejecución de la primera consulta es mayor, y el resto se mantienen estables en torno a 0,1 segundos. Esta consulta es de agregación, sin embargo, los resultados son muy homogéneos, esto puede deberse al poco tráfico en el Estrecho en las ventanas temporales especificadas.

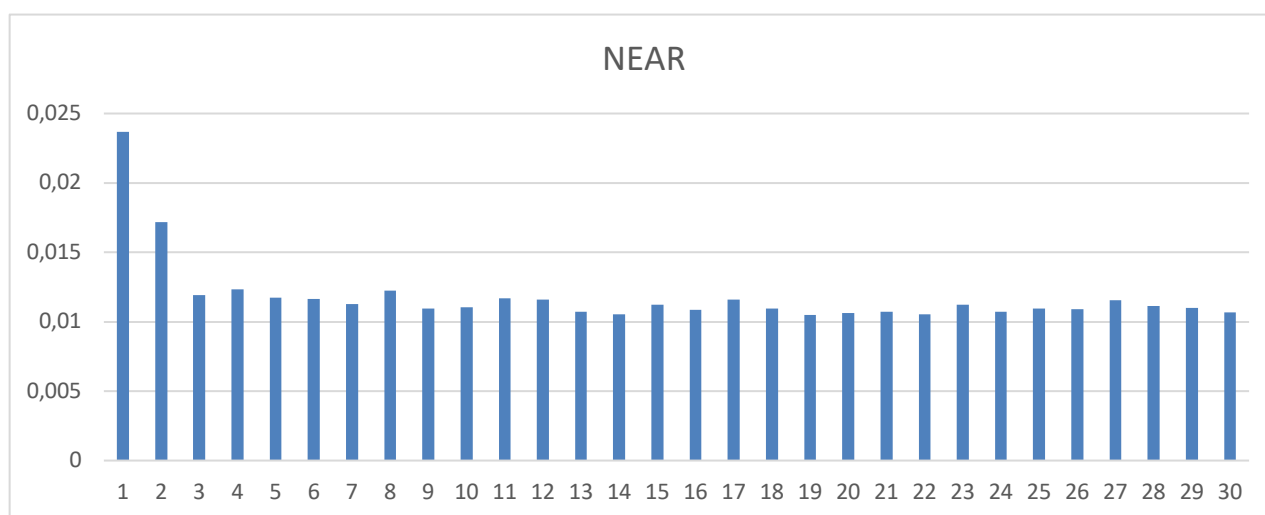


Figura 4-9 Diagrama de barras consultas NEAR

4.1.3.9 Consulta ZONE

Al ejecutar el *script* “searchZone.py” obtenemos el tiempo de ejecución en buscar si hay tramas AIS en un polígono creado por las coordenadas especificadas en la consulta. En este caso se han hecho consultas en un polígono creado en el Estrecho de Gibraltar y otro en el Dispositivo de Separación del Tráfico de Finisterre.

La consulta utilizada en el *script* es la siguiente:

Las primeras 15 consultas se han realizado en el Estrecho de Gibraltar.

```
{"query": {"bool": {"must": {"match_all": {}}},
"filter": {"geo_polygon": {"position": {"points": [{"lat": 36.09, "lon": -5.26}, {"lat": 35.94, "lon": -5.28}, {"lat": 36.03, "lon": -5.93}, {"lat": 35.83, "lon": -5.95}]}}}},
"size": _size}
```

La segunda quincena en el Dispositivo de Separación del Tráfico de Finisterre.

```
{"query": {"bool": {"must": {"match_all": {}}},
"filter": {"geo_polygon": {"position": {"points": [{"lat": 43.32, "lon": -10.05}, {"lat": 43.19, "lon": -10.14}, {"lat": 42.53, "lon": -10.13}, {"lat": 43.20, "lon": -9.36}, {"lat": 43.10, "lon": -9.44}, {"lat": 42.52, "lon": -9.43}]}}}},
"size": _size}
```

Al ejecutar el script como:

```
python searchZone.py
```

Nos da una respuesta como la que se indica a continuación:

GE0,0.028567,2432 (Estrecho de Gibraltar)

GE0,0.034344,1382 (DST Finisterre)

El resultado indica que ha habido 2432 tramas AIS en el Estrecho de Gibraltar y 1382 en el DST de Finisterre.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el script 30 veces. Las 15 primeras consultas son en el Estrecho de Gibraltar y las siguientes son en el DST de Finisterre.

MEDIA	0,02791573
DESVIACIÓN TÍPICA	0,005008
MEDIANA	0,028481
VALOR MÁXIMO	0,048252
VALOR MÍNIMO	0,020379

Tabla 14 Estadísticas consultas ZONE

En la Figura 4-10 se puede observar como la primera consulta requiere un mayor tiempo de ejecución, y el resto se mantienen estables en torno a 0,03 segundos.

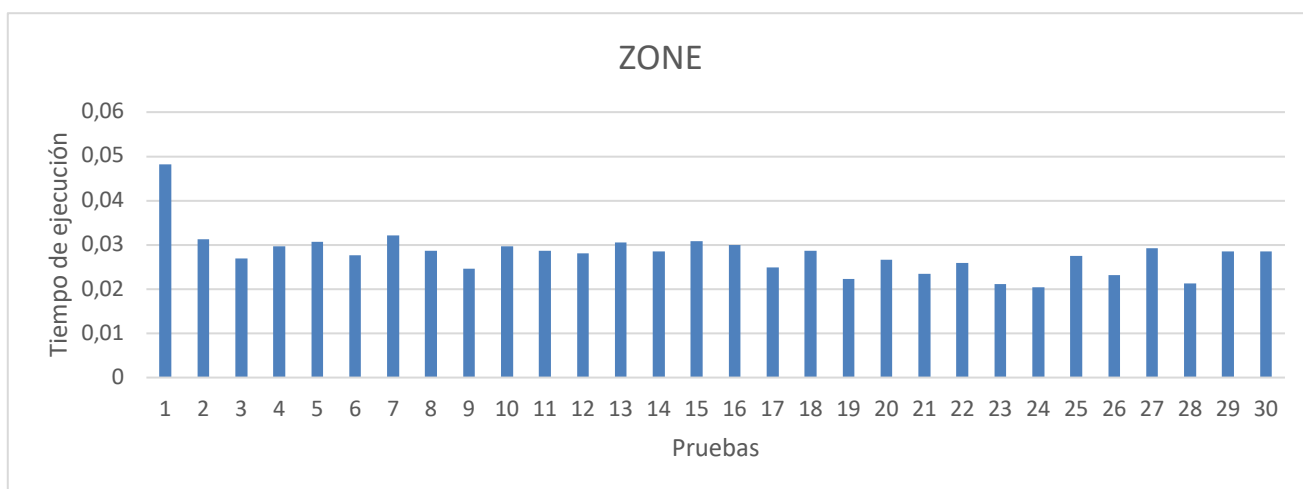


Figura 4-10 Diagrama de barras consultas ZONE

4.1.3.10 Consulta NEARGEO

Al ejecutar el *script* “searchNearGeo.py” obtenemos el tiempo de ejecución en buscar si hay tramas AIS que estén próximas en el tiempo/espacio a otra dado su MMSI y sello temporal. La consulta se realiza dados un rango de tiempo de 30 minutos centrado en el instante temporal deseado, y una distancia en kilómetros centrada en la posición en la que se emitió la trama AIS de interés.

En este caso hay que realizar dos consultas. La primera de ellas a partir del MMSI y el sello temporal especificado, con lo que se obtiene la posición del barco de interés en el instante deseado. Con la posición obtenida y el tercer parámetro, que es la distancia, se realiza una segunda consulta para obtener las tramas AIS emitidas en ese contexto espaciotemporal. Las consultas utilizadas en el *script* son las siguientes:

```
# Realizamos una consulta para obtener la posición del barco de interés en el instante de tiempo deseado.
```

```
{ "query": { "bool": { "must": [ { "match": { "mmsi": sys.argv[1] } },
{ "match": { "@timestamp": sys.argv[2] } } ] } },
"size": _size }
```

```
# Realizamos una segunda consulta para encontrar los barcos que estén a una determinada distancia radial de la posición de interés.
```

```
{ "query": { "bool": { "must": { "range": { "@timestamp": { "gte": _init_time,
"lte": _end_time } } },
"filter": { "geo_distance": { "distance": sys.argv[3], "position": _position } } },
"size": _size }
```

Al ejecutar el script como:

```
python searchNearGeo.py 257659000 1613291914000 13km
```

Nos da una respuesta como la que se indica a continuación:

```
NEARGEO,0.021204,21,257659000,1613291914000,13km
```

El resultado indica que hay 21 tramas AIS en un radio de 13 km centrado en la trama AIS del barco de MMSI 257659000 en un intervalo de tiempo de 15 minutos antes y después del instante de tiempo especificado, 1613291914000.

En la siguiente tabla se muestran una serie de estadísticas del tiempo de ejecución (medido en segundos) obtenidas tras ejecutar el script 30 veces. Se han introducido MMSI aleatorios y tiempos acordes al rango de 30 minutos para poder obtener resultados, ya que, si no hay trama AIS en ese rango de tiempo, no se recibe respuesta. Las distancias medidas a partir de la posición de interés también se han escogido de manera aleatoria entre 1 y 20 km.

MEDIA	0,0237665
DESVIACIÓN TÍPICA	0,00709031
MEDIANA	0,0207925
VALOR MÁXIMO	0,040367
VALOR MÍNIMO	0,01636

Tabla 15 Estadísticas consultas NEARGEO

En la Figura 4-11 se puede observar como las primeras consultas requieren mayor tiempo de ejecución.

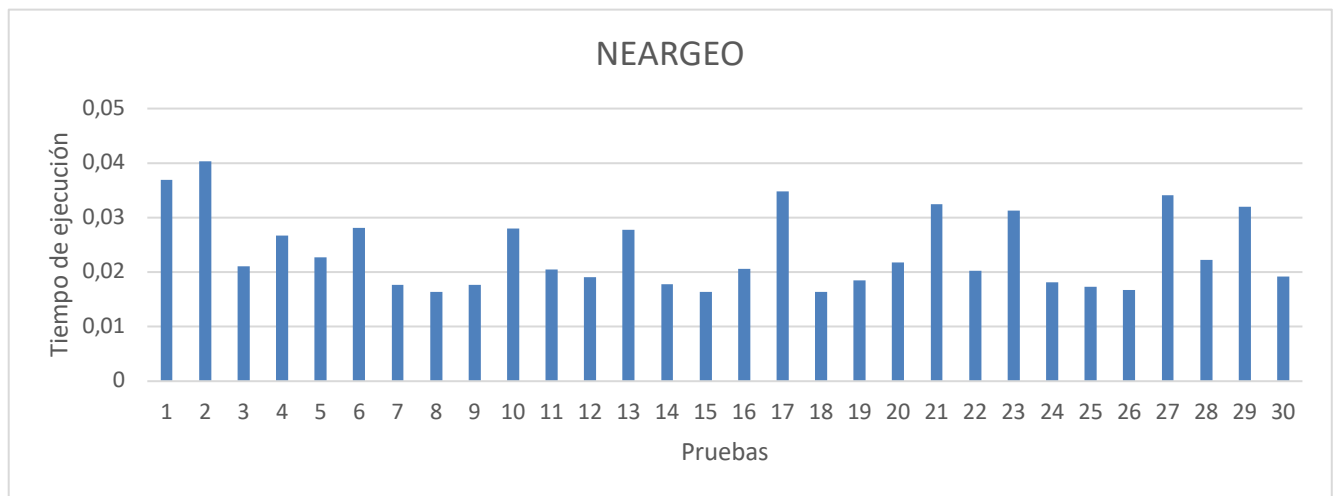


Figura 4-11 Diagrama de barras consultas NEARGEO

4.1.4 Resumen de los resultados

En este apartado se incluye una tabla a modo de resumen de los resultados obtenidos (en segundos) en las diferentes pruebas realizadas.

CONSULTA	VALOR MINIMO	MEDIANA	MEDIA	DESVIACIÓN TÍPICA	VALOR MÁXIMO
ALL	0,008464	0,0124715	0,01249103	0,00259358	0,023125
PORT	0,021369	0,0241095	0,02510865	0,00665991	0,059497
LENGTH	0,025195	0,100046	0,10385057	0,06508903	0,025195
POS	0,012301	0,024719	0,02337813	0,00455619	0,031733
CELL	0,007212	0,0242055	0,04977517	0,06039915	0,216762
TIME	0,008227	0,0100065	0,01039857	0,00215862	0,021083
MMSI	0,008666	0,0197235	0,0191374	0,00437535	0,02582
NEAR	0,010468	0,0111025	0,0117888	0,00254309	0,023663
ZONE	0,020379	0,028481	0,02791573	0,005008	0,048252
NEARGEO	0,01636	0,0207925	0,0237665	0,00709031	0,040367

Tabla 16 Resumen de los resultados

Tras estudiar las estadísticas de los tiempos obtenidos en las diferentes consultas se pueden obtener una serie de conclusiones:

- Las consultas con agregaciones sufren variaciones en el tiempo de procesamiento en función del número de resultados obtenidos. Por ello en las consultas CELL y LENGTH se registran

los mayores tiempos de respuesta (entre 50 y 100 ms). La consulta NEAR lleva una agregación también, sin embargo, debido a la poca cantidad de resultados en la zona de consulta, los tiempos de respuesta son bajos.

- En el resto de consultas los tiempos oscilan por debajo de los 30 ms.
- Cabe destacar la rapidez con la que se pueden llegar a realizar consultas para buscar tramas AIS con un determinado MMSI (19 ms) o en un rango de tiempo determinado (10 ms).
- Elasticsearch nos permite obtener información sobre todos los barcos que se encuentren en el Estrecho de Gibraltar o en el DST de Finisterre en tan solo 28 ms.
- Podemos estudiar rápidamente el entorno de un barco de interés en cualquier instante temporal en solo 23 ms.
- Es posible conocer la ruta de un barco a través de sus últimas emisiones AIS y los puertos en los que ha estado en 25 ms.

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se presentan las principales conclusiones extraídas de la realización del trabajo, así como posibles líneas futuras de desarrollo.

5.1 Conclusiones

En este TFG se ha realizado un estudio de algunas de las herramientas Big data de código libre más utilizadas en la actualidad para almacenamiento de flujos de información. Posteriormente, se ha llevado a cabo una comparación cualitativa entre ellas, eligiendo finalmente Elasticsearch. Tras haber instalado y probado la herramienta, de los resultados obtenidos se pueden extraer una serie de conclusiones que nos hacen ver la importancia del empleo de este tipo de plataformas para la Armada.:

1. La cantidad de datos en el entorno marítimo está creciendo de manera exponencial, sin precedentes. Es por ello que se hace necesario el uso de herramientas Big data, tales como las que se han presentado en este TFG, para poder almacenar y gestionar el flujo continuo de tramas AIS de una manera rápida y efectiva.
2. Este tipo de herramientas aporta una gran utilidad a los propósitos de la Armada. El entorno marítimo es un medio difícil de controlar, debido a su inmensidad y a la cantidad de barcos que se hacen a la mar hoy en día para todo tipo de propósitos. El uso de herramientas Big data ayuda a alcanzar el conocimiento del entorno marítimo, que es uno de los cometidos principales de la Armada en el ámbito de la Vigilancia y Seguridad Marítima y por lo tanto en la Seguridad Nacional.
3. Estas plataformas permitirían al COVAM detectar anomalías en la mar y actividades potencialmente ilícitas. De manera rápida y efectiva, mediante búsquedas automatizadas, es posible realizar un seguimiento a un barco a través de sus últimas posiciones GPS o los puertos en los que ha recalado. Podemos buscar dónde se encuentra un barco en el instante deseado, si está parado o si está cerca de otro barco de interés. Estudiando los resultados de las búsquedas sería posible determinar si se están llevando a cabo actividades sospechosas o ilegales en la mar.
4. Por limitaciones temporales no se han podido probar todas las herramientas para comparar los resultados obtenidos. Sin embargo, Elasticsearch ha demostrado tener unas funcionalidades y un rendimiento adecuados para trabajar con datos relativos a tramas AIS, gracias a su elevada velocidad de respuesta ante cualquier tipo de consulta. Se han realizado 10 tipos de consultas diferentes en un índice con 5 millones de tramas AIS, y todas ellas han dado respuesta con un tiempo de ejecución del orden de milisegundos.

5.2 Líneas futuras

Sin duda, las herramientas de Big data deben enfrentarse al reto de manejar progresivamente un volumen creciente de datos, y hacerlo de una forma eficaz, para lo cual se debe avanzar en el diseño de aplicaciones software, que indudablemente deben ir de la mano de un hardware con mejores prestaciones de almacenamiento y procesamiento de datos.

Una vez que han sido definidos mecanismos para efectuar determinadas búsquedas, sería de utilidad que el sistema fuese capaz de utilizarlos como apoyo a la hora de detectar anomalías, aprender de ellas y relacionarlas con otras. Por ejemplo, si en una determinada zona se observan anomalías cinemáticas, se podrían relacionar con el tipo de barcos implicados. Para ello se podrían aplicar técnicas de inteligencia artificial a los datos obtenidos de las búsquedas.

La conexión entre las herramientas de Big data y el usuario debe cuidarse en extremo, de tal forma que tanto la programación de búsquedas, como la explotación de los datos obtenidos, sean accesibles sin la necesidad de conocimientos avanzados.

No debe olvidarse que la vigilancia marítima basada en el análisis de datos AIS debe enfrentarse a su talón de Aquiles, que no es más que la posibilidad relativamente sencilla de que estos datos AIS sean manipulados. Estas manipulaciones pueden ser básicas, como la alteración malintencionada de los datos propios, interferencias en el espectro electromagnético, para anular o perturbar las emisiones de los buques, u otras más sofisticadas que permita crear trazas ficticias o atacar bases de datos de los centros de vigilancia. La creación de algoritmos que pudieran detectar estas anomalías constituiría una herramienta de importancia capital para luchar contra esta amenaza.

Otra posible línea futura podría ser probar el rendimiento y capacidades de consulta de otras herramientas potencialmente útiles, como InfluxDB, para poder hacer una comparación cuantitativa y no solo cualitativa entre ellas.

6 BIBLIOGRAFÍA

- [1] I. Lytra, M.E. Vidal, F. Orlandi, J. Attard, «A Big Data Architecture for Managing Oceans of Data and Maritime Applications,» 2017. [En línea]. Available: https://www.researchgate.net/publication/322997369_A_big_data_architecture_for_managing_oceans_of_data_and_maritime_applications. [Último acceso: 15 enero 2021].
- [2] S. Bhattacharjee, «Automatic Identification System (AIS): Integrating and Identifying Marine Communication Channels,» 25 Diciembre 2020. [En línea]. Available: <https://www.marineinsight.com/marine-navigation/automatic-identification-system-ais-integrating-and-identifying-marine-communication-channels/>. [Último acceso: 15 enero 2021].
- [3] D. Yacht, «Entendiendo el sistema AIS,» [En línea]. Available: <https://digitalyacht.es/wp-content/uploads/2018/06/AIS-Training-in-Spanish.pdf>. [Último acceso: 15 enero 2021].
- [4] J. L. Urcelay, Interviewee, *La transformación digital y el sostenimiento 4.0 en la Armada*. [Entrevista]. 4 Marzo 2019.
- [5] DGAM, «DDV Conocimiento del entorno marítimo [Informe],» 2017.
- [6] «INETUM Positive digital flow,» [En línea]. Available: <https://www.inetum.com.es/es/index.html>. [Último acceso: 3 febrero 2021].
- [7] Á. Robledano, «Qué es MongoDB,» 28 Octubre 2019. [En línea]. Available: <https://openwebinars.net/blog/que-es-mongodb/>. [Último acceso: 17 enero 2021].
- [8] R. Carrasco, «El núcleo y la arquitectura de MongoDB,» [En línea]. Available: <https://www.ramoncarrasco.es/es/content/es/kb/129/el-nucleo-mongodb-y-la-arquitectura-de-mongodb>. [Último acceso: 18 enero 2021].
- [9] M. Altarade, «Bases de datos NoSQL : Guía definitiva,» 20 Abril 2017. [En línea]. Available: <https://pandorafms.com/blog/es/bases-de-datos-nosql/>. [Último acceso: 18 enero 2021].
- [10] J. Drumgoole, «SEGA HARDlight Migrates to MongoDB Atlas to Simplify Ops and Improve Experience for Millions of Mobile Gamers,» 15 Marzo 2018. [En línea]. Available:

- <https://www.mongodb.com/blog/post/sega-hardlight-migrates-to-mongodb-atlas-simplify-ops-improve-experience-mobile-gamers>. [Último acceso: 19 enero 2021].
- [11] «InfluxDB: Explicación, ventajas y primeros pasos,» 28 Septiembre 2020. [En línea]. Available: <https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/que-es-influxdb/>. [Último acceso: 19 enero 2021].
- [12] «DB-Engines Ranking - Trend of Time Series DBMS Popularity,» [En línea]. Available: https://db-engines.com/en/ranking_trend/time+series+dbms. [Último acceso: 19 enero 2021].
- [13] M. Hernández, «TICK Stack para BigData con series temporales,» 4 Julio 2018. [En línea]. Available: <https://www.adictosaltrabajo.com/2018/07/04/tick-stack-para-bigdata-con-series-temporales/>. [Último acceso: 20 enero 2021].
- [14] «Paypal Influxdata,» [En línea]. Available: <https://www.influxdata.com/customer/paypal/>. [Último acceso: 20 enero 2021].
- [15] C. Churilo, «Building a Scalable End-to-End Host Monitoring Solution with InfluxDB Enterprise,» 2020 Enero 2020. [En línea]. Available: <https://www.influxdata.com/blog/building-a-scalable-end-to-end-host-monitoring-solution-with-influxdb-enterprise/>. [Último acceso: 20 enero 2021].
- [16] «Introducción a Kafka Streams y KSQL,» 2 Enero 2021. [En línea]. Available: <https://aprenderbigdata.com/kafka-streams-ksql/>. [Último acceso: 21 enero 2021].
- [17] «An overview of ksqlDB,» [En línea]. Available: <https://ksqldb.io/overview.html>. [Último acceso: 21 enero 2021].
- [18] F. Lora, «KAFKA y KSQL: la combinación perfecta para el streaming de datos,» 25 Junio 2020. [En línea]. Available: <https://www.viewnext.com/kafka-y-ksql-la-combinacion-perfecta-para-el-streaming-de-datos/>. [Último acceso: 21 enero 2021].
- [19] J. Kreps, «Introducing ksqlDB,» 20 Noviembre 2019. [En línea]. Available: <https://www.confluent.io/blog/intro-to-ksqldb-sql-database-streaming/>. [Último acceso: 22 enero 2021].
- [20] «RocksDB,» [En línea]. Available: <https://rocksdb.org> RocksDB. [Último acceso: 22 enero 2021].
- [21] «Bosch Power Tools Drives a Digital Transformation and Smart Tools Innovation by Streaming IoT Data with Confluent Cloud,» [En línea]. Available: https://assets.confluent.io/m/47b5f3952ebbc5a4/original/20190919-CS-Bosch.pdf?_ga=2.15567731.418037092.1611216822-1971221883.1611216822. [Último acceso: 22 enero 2021].
- [22] «What is Elasticsearch?,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/7.10/elasticsearch-intro.html>. [Último acceso: 24 enero 2021].
- [23] P. S. González, «Características de la arquitectura de Elasticsearch,» 19 febrero 2019. [En línea]. Available: <https://openwebinars.net/blog/caracteristicas-de-la-arquitectura-de-elasticsearch/>. [Último acceso: 24 enero 2021].
- [24] «DB-Engines Ranking - Trend of Search Engines Popularity,» [En línea]. Available: https://db-engines.com/en/ranking_trend/search+engine. [Último acceso: 23 enero 2021].

- [25] «DB-Engines,» [En línea]. Available: https://db-engines.com/en/ranking_trend/search+engine. [Último acceso: 24 enero 2021].
- [26] «Data in: documents and indices,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/7.10/documents-indices.html>. [Último acceso: 24 enero 2021].
- [27] «Information out: search and analyze,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/7.10/search-analyze.html>. [Último acceso: 24 enero 2021].
- [28] «Scalability and resilience: clusters, nodes, and shards,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/7.10/scalability.html>. [Último acceso: 24 enero 2021].
- [29] C. Wang, «Lenovo: A Unified View of Sales Across China through Log Analysis,» 11 marzo 2019. [En línea]. Available: <https://www.elastic.co/es/blog/lenovo-a-unified-view-of-sales-across-china-through-log-analysis>. [Último acceso: 24 enero 2021].
- [30] Á. Robledano, «Qué es MySQL: Características y ventajas,» 24 noviembre 2019. [En línea]. Available: <https://openwebinars.net/blog/que-es-mysql/>. [Último acceso: 25 enero 2021].
- [31] «What is MySQL?,» [En línea]. Available: <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>. [Último acceso: 25 enero 2021].
- [32] «DB-Engines Ranking - Trend Popularity,» [En línea]. Available: https://db-engines.com/en/ranking_trend. [Último acceso: 25 enero 2021].
- [33] M. Á. Sicilia, «Visión general de la arquitectura de MySQL 5.1.,» [En línea]. Available: <https://cnx.org/contents/-jPvAviC@1/Visión-general-de-la-arquitectura-de-MySQL-5-1>. [Último acceso: 26 enero 2021].
- [34] S. Vaovlic, R. Pilani, S. Parulekar, S. Handa, «The Netflix Tech Blog,» 21 junio 2016. [En línea]. Available: <https://netflixtechblog.com/netflix-billing-migration-to-aws-451fba085a4>. [Último acceso: 25 enero 2021].
- [35] «Apache Kylin,» [En línea]. Available: <http://kylin.apache.org>. [Último acceso: 29 enero 2021].
- [36] «DB-Engines Ranking - Trend of Relational DBMS Popularity,» [En línea]. Available: https://db-engines.com/en/ranking_trend/relational+dbms. [Último acceso: 2 marzo 2021].
- [37] M. Á. Sotomayor, «Apache Kylin: cómo analizar Terabytes en menos de un segundo,» 8 septiembre 2020. [En línea]. Available: <https://blog.gft.com/es/2020/09/08/apache-kylin-como-analizar-terabytes-en-menos-de-un-segundo/>. [Último acceso: 29 enero 2021].
- [38] «Application of Apache Kylin in Meituan Dianping,» [En línea]. Available: <https://programmersought.com/article/80804430469/>. [Último acceso: 29 enero 2021].
- [39] «Overview Apache Cassandra,» 2016. [En línea]. Available: <https://cassandra.apache.org/doc/latest/architecture/overview.html>. [Último acceso: 30 enero 2021].

- [40] A. R. Mesa, «Qué es Apache Cassandra,» 17 junio 2019. [En línea]. Available: <https://openwebinars.net/blog/que-es-apache-cassandra/>. [Último acceso: 30 enero 2021].
- [41] «DB-Engines Ranking - Trend of Wide Column Stores Popularity,» [En línea]. Available: https://db-engines.com/en/ranking_trend/wide+column+store. [Último acceso: 2 marzo 2021].
- [42] J. A. Alfaro, «Cassandra, funcionamiento y principios,» 9 diciembre 2019. [En línea]. Available: <https://medium.com/@jarrietaalfaro/cassandra-funcionamiento-y-principios-926d5bebe576>. [Último acceso: 30 enero 2021].
- [43] A. Whang, «Accelerate Rewind: How Instagram Uses Cassandra to Operate on a Global Scale,» 2 abril 2020. [En línea]. Available: <https://www.datastax.com/blog/accelerate-rewind-how-instagram-uses-cassandra-operate-global-scale-2>. [Último acceso: 30 enero 2021].
- [44] «Introduction to Redis,» [En línea]. Available: <https://redis.io/topics/introduction>. [Último acceso: 30 enero 2021].
- [45] «DB-Engines Ranking - Trend of Key-value Stores Popularity,» [En línea]. Available: https://db-engines.com/en/ranking_trend/key-value+store. [Último acceso: 2 marzo 2021].
- [46] «Amazon ElastiCache. Redis,» [En línea]. Available: <https://aws.amazon.com/es/redis/>. [Último acceso: 30 enero 2021].
- [47] «Bases de datos documentales: ¿cómo funciona un document store?,» 23 marzo 2020. [En línea]. Available: <https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/bases-de-datos-documentales/>. [Último acceso: 2 marzo 2021].
- [48] «Bases de datos NoSQL 3: Grafos, series temporales y repositorios de contenido,» 21 noviembre 2018. [En línea]. Available: <https://bosonit.com/nosql-3-databases-graphs-time-series-and-content-repositories/>. [Último acceso: 2 marzo 2021].
- [49] «What Is a Search-Engine Database?,» [En línea]. Available: <https://aws.amazon.com/es/nosql/search/>. [Último acceso: 2 marzo 2021].
- [50] «Bases de datos relacional ¿Qué es y sus características?,» [En línea]. Available: <https://ayudaleyprotecciondatos.es/bases-de-datos/relacional/>. [Último acceso: 2 marzo 2021].
- [51] «NoSQL wide-column stores demystified,» 15 Diciembre 2020. [En línea]. Available: <https://blog.logrocket.com/nosql-wide-column-stores-demystified/>. [Último acceso: 2 marzo 2021].
- [52] «Cubos OLAP,» [En línea]. Available: https://www.ecured.cu/Cubos_OLAP. [Último acceso: 2 marzo 2021].
- [53] «Bases de datos clave-valor,» 10 marzo 2020. [En línea]. Available: <https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/bases-de-datos-clave-valor/>. [Último acceso: 2 marzo 2021].
- [54] «SQL to MongoDB Mapping Chart,» [En línea]. Available: <https://docs.mongodb.com/manual/reference/sql-comparison/>. [Último acceso: 1 marzo 2021].

- [55] «Get started with Flux,» [En línea]. Available: <https://docs.influxdata.com/influxdb/v2.0/query-data/get-started/>. [Último acceso: 1 marzo 2021].
- [56] «The SQL Language. KsqlDB,» [En línea]. Available: <https://docs.ksqldb.io/en/latest/reference/sql/syntax/lexical-structure/>. [Último acceso: 1 marzo 2021].
- [57] «Query DSL Elasticsearch,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html>. [Último acceso: 1 marzo 2021].
- [58] «SQL Statements. MySQL,» [En línea]. Available: <https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html>. [Último acceso: 1 marzo 2021].
- [59] «SQL Reference. Apache Kylin,» [En línea]. Available: http://kylin.apache.org/docs/tutorial/sql_reference.html. [Último acceso: 1 marzo 2021].
- [60] «The Cassandra Query Language (CQL),» [En línea]. Available: <https://cassandra.apache.org/doc/latest/cql/index.html>. [Último acceso: 1 marzo 2021].
- [61] «Commands Redis,» [En línea]. Available: <https://redis.io/commands>. [Último acceso: 1 marzo 2021].
- [62] «MongoDB dashboard for Grafana,» [En línea]. Available: <https://grafana.com/grafana/dashboards/2583>. [Último acceso: 28 febrero 2021].
- [63] «Using InfluxDB in Grafana,» [En línea]. Available: <https://grafana.com/docs/grafana/latest/datasources/influxdb/>. [Último acceso: 28 febrero 2021].
- [64] S. Bhat, K. Waehner, «Real-Time Analytics and Monitoring Dashboards with Apache Kafka and Rockset,» 26 septiembre 2019. [En línea]. Available: <https://www.confluent.io/blog/analytics-with-apache-kafka-and-rockset/>. [Último acceso: 1 marzo 2021].
- [65] «Kibana Guide,» [En línea]. Available: <https://www.elastic.co/guide/en/kibana/current/index.html>. [Último acceso: 1 marzo 2021].
- [66] «Using MySQL in Grafana,» [En línea]. Available: <https://grafana.com/docs/grafana/latest/datasources/mysql/>. [Último acceso: 1 marzo 2021].
- [67] «Cassandra Dashboard for Grafana,» [En línea]. Available: <https://grafana.com/grafana/dashboards/5408>. [Último acceso: 1 marzo 2021].
- [68] «Redis plugin for Grafana,» [En línea]. Available: <https://grafana.com/grafana/plugins/redis-datasource>. [Último acceso: 1 marzo 2021].
- [69] «Storm MongoDB integration,» [En línea]. Available: <https://storm.apache.org/releases/2.1.0/storm-mongodb.html>. [Último acceso: 28 febrero 2021].
- [70] «Storm-metrics-InfluxDB,» [En línea]. Available: <https://github.com/christiangda/storm-metrics-influxdb>. [Último acceso: 28 febrero 2021].

- [71] «Storm Elasticsearch Integration,» [En línea]. Available: <https://storm.apache.org/releases/current/storm-elasticsearch.html>. [Último acceso: 1 marzo 2021].
- [72] «Storm JDBC Integration,» [En línea]. Available: <https://storm.apache.org/releases/1.2.3/storm-jdbc.html>. [Último acceso: 1 marzo 2021].
- [73] «Storm Cassandra Integration,» [En línea]. Available: <http://storm.apache.org/releases/current/storm-cassandra.html>. [Último acceso: 1 marzo 2021].
- [74] «Storm Redis Integration,» [En línea]. Available: <https://storm.apache.org/releases/2.0.0/storm-redis.html>. [Último acceso: 1 marzo 2021].
- [75] «Create index MongoDB,» [En línea]. Available: <https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/>. [Último acceso: 28 febrero 2021].
- [76] «Write data with the InfluxDB API,» [En línea]. Available: https://docs.influxdata.com/influxdb/v1.8/guides/write_data/. [Último acceso: 28 febrero 2021].
- [77] «Create Table ksqldb,» [En línea]. Available: <https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/create-table/>. [Último acceso: 28 febrero 2012].
- [78] «Create index Elasticsearch,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/6.8/indices-create-index.html>. [Último acceso: 1 marzo 2021].
- [79] «Create index statement MySQL,» [En línea]. Available: <https://dev.mysql.com/doc/refman/8.0/en/create-index.html>. [Último acceso: 1 marzo 2021].
- [80] «Real-time OLAP,» [En línea]. Available: http://kylin.apache.org/docs/tutorial/realtime_olap.html. [Último acceso: 2 marzo 2021].
- [81] «Create Index Cassandra,» [En línea]. Available: https://docs.datastax.com/en/cql-oss/3.x/cql/cql_reference/cqlCreateIndex.html. [Último acceso: 1 marzo 2021].
- [82] «Indexes Redis,» [En línea]. Available: <https://redis.io/topics/indexes>. [Último acceso: 1 marzo 2021].
- [83] «Find objects between two dates in MongoDB,» [En línea]. Available: <https://www.tutorialspoint.com/find-objects-between-two-dates-in-mongodb>. [Último acceso: 28 febrero 2021].
- [84] «InfluxDB Timestamp and Query Time Range,» [En línea]. Available: <https://www.influxdata.com/blog/tldr-influxdb-tech-tips-november-10-2016/>. [Último acceso: 28 febrero 2021].
- [85] «Time and Windows. Ksqldb,» [En línea]. Available: <https://docs.ksqldb.io/en/latest/concepts/time-and-windows-in-ksqldb-queries/>. [Último acceso: 28 febrero 2021].
- [86] «Range query Elasticsearch,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-range-query.html>. [Último acceso: 1 marzo 2021].

- [87] «The DATE, DATETIME, and TIMESTAMP Types. MySQL,» [En línea]. Available: <https://dev.mysql.com/doc/refman/8.0/en/datetime.html>. [Último acceso: 1 marzo 2021].
- [88] «RANGE Partitioning MySQL,» [En línea]. Available: <https://dev.mysql.com/doc/refman/8.0/en/partitioning-range.html>. [Último acceso: 1 marzo 2021].
- [89] W. Liang, H. Wang, M.E. Orłowska, «Range queries in dynamic OLAP data cubes,» Australia, 2000.
- [90] «Using date ranges in solr_query. Apache Cassandra,» [En línea]. Available: https://docs.datastax.com/en/dse/6.0/cql/cql/cql_using/search_index/dateRange.html. [Último acceso: 1 marzo 2021].
- [91] «Sorted set time series. Redis,» [En línea]. Available: <https://redislabs.com/redis-best-practices/time-series/sorted-set-time-series/>.
- [92] «Geospatial Query MongoDB,» [En línea]. Available: <https://docs.mongodb.com/manual/geospatial-queries/>. [Último acceso: 28 febrero 2021].
- [93] «Work with geo-temporal data. InfluxDB,» [En línea]. Available: <https://docs.influxdata.com/influxdb/cloud/query-data/flux/geo/>. [Último acceso: 28 febrero 2021].
- [94] «Scalar functions. KsqlDB,» [En línea]. Available: https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/scalar-functions/#geo_distance. [Último acceso: 28 febrero 2021].
- [95] «Building a Materialized Cache with ksqlDB,» [En línea]. Available: <https://www.confluent.io/blog/build-materialized-cache-with-ksqldb/>. [Último acceso: 28 febrero 2021].
- [96] «Location Analytics - Real-Time Geofencing using Apache Kafka,» [En línea]. Available: <https://www.slideshare.net/gschmutz/location-analytics-realtime-geofencing-using-apache-kafka-203115651>. [Último acceso: 28 febrero 2021].
- [97] «Geo queries Elasticsearch,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-queries.html>. [Último acceso: 1 marzo 2021].
- [98] «Spatial Data Types. MySQL,» [En línea]. Available: <https://dev.mysql.com/doc/refman/8.0/en/spatial-types.html>. [Último acceso: 1 marzo 2021].
- [99] S. Bimonte, M. Zaamoune, P. Beaune, «Conceptual design and implementation of spatial data ware- houses integrating regular grids of points,» *International Journal of Digital Earth*, vol. 10, pp. 901-902, 2016.
- [100] «Geospatial queries for Point and LineString. Apache Cassandra,» [En línea]. Available: https://docs.datastax.com/en/dse/6.8/cql/cql/cql_using/search_index/queriesGeoSpatial.html. [Último acceso: 1 marzo 2021].
- [101] «Geospatial Redis,» [En línea]. Available: <https://redislabs.com/redis-best-practices/indexing-patterns/geospatial/>. [Último acceso: 1 marzo 2021].

- [102] «Aggregation MongoDB,» [En línea]. Available: <https://docs.mongodb.com/manual/aggregation/>. [Último acceso: 28 febrero 2021].
- [103] «Flux built-in aggregate transformations,» [En línea]. Available: <https://docs.influxdata.com/influxdb/v2.0/reference/flux/stdlib/built-in/transformations/aggregates/>. [Último acceso: 28 febrero 2021].
- [104] «Aggregation functions KsqlDB,» [En línea]. Available: <https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/aggregate-functions/>. [Último acceso: 28 febrero 2021].
- [105] «Aggregations Elasticsearch,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html>. [Último acceso: 1 marzo 2021].
- [106] «Aggregate Function Descriptions MySQL,» [En línea]. Available: <https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html>. [Último acceso: 1 marzo 2021].
- [107] «OLAP expression syntax reference,» [En línea]. Available: https://docs.oracle.com/database/121/OLAXS/olap_functions.htm#OLAXS169. [Último acceso: 1 marzo 2021].
- [108] «Create Aggregate Cassandra,» [En línea]. Available: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateAggregate.html. [Último acceso: 1 marzo 2021].
- [109] «Aggregating users by location. Redis,» [En línea]. Available: <https://redislabs.com/ebook/part-2-core-concepts/chapter-6-application-components-in-redis/6-6-distributing-files-with-redis/6-6-1-aggregating-users-by-location/>. [Último acceso: 1 marzo 2021].
- [110] Q. Brunet, «InfluxDB vs Elasticsearch for time series and metrics data,» 9 mayo 2019. [En línea]. Available: <https://jolicode.com/blog/influxdb-vs-elasticsearch-for-time-series-and-metrics-data>. [Último acceso: 28 febrero 2021].
- [111] «Elastic,» [En línea]. Available: <https://www.elastic.co/es/>. [Último acceso: 14 febrero 2021].
- [112] «Index some documents. Elasticsearch,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started-index.html>. [Último acceso: 2 marzo 2021].
- [113] Secutiry, U.S Department of Homeland, «AIS CLASS A SHIP STATIC AND VOYAGE RELATED DATA (MESSAGE 5),» [En línea]. Available: <https://www.navcen.uscg.gov/?pageName=AIMessagesAStatic>. [Último acceso: 23 febrero 2021].
- [114] I. Brodsky, «H3: Uber's Hexagonal Hierarchical Spatial Index,» 27 junio 2018. [En línea]. Available: <https://eng.uber.com/h3/>. [Último acceso: 23 febrero 2021].
- [115] «Wikipedia la enciclopedia libre,» [En línea]. Available: <https://es.wikipedia.org/wiki/Wikipedia:Portada>. [Último acceso: 23 febrero 2021].
- [116] «1000marcas,» [En línea]. Available: <https://1000marcas.net/bosch-logo/>. [Último acceso: 22 enero 2021].

- [117] «MySQL,» [En línea]. Available: <https://www.mysql.com>. [Último acceso: 25 enero 2021].
- [118] M. Á. Sotomayor, «GFT,» 8 septiembre 2020. [En línea]. Available: <https://blog.gft.com/es/2020/09/08/apache-kylin-como-analizar-terabytes-en-menos-de-un-segundo/>. [Último acceso: 29 enero 2021].
- [119] S. G. Ruisanchez, «Ayse Lucus,» 21 octubre 2020. [En línea]. Available: <https://www.ayselucus.es/noticia/privacidad-en-instagram-el-nuevo-gran-hermano>. [Último acceso: 30 enero 2021].
- [120] «Coventry & Warwickshire Champions,» [En línea]. Available: <https://www.coventry-warwickshire.co.uk/champion/sega-hardlight>. [Último acceso: 19 enero 2021].
- [121] «Twitter,» [En línea]. Available: <https://twitter.com/ksqldb>. [Último acceso: 21 enero 2021].
- [122] «InfluxDB,» [En línea]. Available: <https://www.influxdata.com>. [Último acceso: 19 enero 2021].
- [123] «Python,» [En línea]. Available: <https://www.python.org>. [Último acceso: 20 febrero 2021].
- [124] X. Campaña, «El ascenso y la caída del cubo OLAP,» 22 mayo 2020. [En línea]. Available: <https://www.itconsultors.com/el-ascenso-y-la-caida-del-cubo-olap>. [Último acceso: 29 enero 2021].
- [125] M. Dianping, «Meituan Opens Global Leading Delivery Platform with New Brand Meituan Delivery,» 6 mayo 2019. [En línea]. Available: <https://www.prnewswire.com/news-releases/meituan-opens-global-leading-delivery-platform-with-new-brand-meituan-delivery-300844106.html>. [Último acceso: 29 enero 2021].
- [126] «DB-Engines Ranking - Trend of Document Stores Popularity,» [En línea]. Available: https://db-engines.com/en/ranking_trend/document+store. [Último acceso: 17 enero 2021].
- [127] D. Llanes, «Instalar y configurar MongoDB en Ubuntu 16.04,» [En línea]. Available: <https://www.agiliacenter.com/instalar-configurar-mongodb-ubuntu/>. [Último acceso: 18 enero 2021].
- [128] L. Miranda, «Netflix Logo,» 3 abril 2020. [En línea]. Available: <https://logos-marcas.com/netflix-logo/>. [Último acceso: 26 enero 2021].
- [129] P. Sheban, «Cassandra Architecture Fundamentals,» 2019 noviembre 2019. [En línea]. Available: <https://medium.com/@pramodshehan/cassandra-architecture-fundamentals-cc617f12b957>. [Último acceso: 30 enero 2021].
- [130] «Transpondedor AIS. Automatic Identification System,» [En línea]. Available: <https://www.unioviedo.es/ate/manuel/ESMC-EyE-Bolonia/TG-AIS.pdf>. [Último acceso: 15 enero 2021].
- [131] «Use Dashboard. Apache Kylin,» [En línea]. Available: http://kylin.apache.org/docs/tutorial/use_dashboard.html. [Último acceso: 28 marzo 2021].

- [132] «Cassandra Logo,» 14 marzo 2013. [En línea]. Available: https://commons.wikimedia.org/wiki/File:Cassandra_logo.svg. [Último acceso: 30 enero 2021].
- [133] «WorldVectorLogo,» [En línea]. Available: <https://worldvectorlogo.com/es/logo/mysql-5>. [Último acceso: 25 enero 2021].
- [134] «Logo Redis,» [En línea]. Available: https://yosoy.dev/redis-que-como-ejemplos/redis_logo/. [Último acceso: 30 enero 2021].
- [135] «Start Searching Elasticsearch,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started-search.html>. [Último acceso: 2 marzo 2021].
- [136] «Create table Apache Cassandra,» [En línea]. Available: <https://cassandra.apache.org/doc/latest/cql/ddl.html#create-table>. [Último acceso: 2 marzo 2021].
- [137] «An introduction to Redis data types and abstractions,» [En línea]. Available: <https://redis.io/topics/data-types-intro>. [Último acceso: 2 marzo 2021].
- [138] «Logotipo Elasticsearch,» 6 octubre 2017. [En línea]. Available: <http://data.cervantesvirtual.com/blog/2017/10/06/introduccion-a-elasticsearch/logotipo-elasticsearch>. [Último acceso: 24 enero 2021].
- [139] «Redis,» [En línea]. Available: <http://www.osenterprise.com.pe/web/redis>. [Último acceso: 30 enero 2021].
- [140] J. Maiz, «Defensa [Artículo de revista],» *Revista Defensa*, n° 420, 8 Septiembre 2017.
- [141] «Logotipo Paypal,» [En línea]. Available: <https://www.paypal.com/es/webapps/mpp/logo-center>. [Último acceso: 20 enero 2021].

ANEXO I: RESULTADOS OBTENIDOS

En las siguientes tablas se reflejan los resultados obtenidos en las diferentes consultas a lo largo de las 30 pruebas. A partir de estos valores de tiempo de ejecución se han obtenido la media, desviación típica, mediana, valor máximo, valor mínimo y los diagramas de barras reflejados en el Capítulo 4.

ALL		
PRUEBAS	TIEMPO DE EJECUCIÓN	RESULTADOS
1	0,023125	10000
2	0,009539	10000
3	0,012229	10000
4	0,00913	10000
5	0,01163	10000
6	0,01159	10000
7	0,011715	10000
8	0,012987	10000
9	0,01671	10000
10	0,011518	10000
11	0,0114	10000
12	0,010509	10000
13	0,008464	10000
14	0,013008	10000
15	0,010164	10000
16	0,011501	10000
17	0,011132	10000
18	0,011209	10000
19	0,011372	10000
20	0,012714	10000
21	0,014339	10000
22	0,01309	10000
23	0,013288	10000
24	0,013312	10000
25	0,013088	10000
26	0,013398	10000
27	0,013146	10000
28	0,013155	10000
29	0,013217	10000
30	0,013052	10000

Tabla 17 Resultados consultas ALL

PORT			
PRUEBAS	TIEMPO DE EJECUCIÓN	RESULTADOS	PUERTO
1	0,059497	4405	VIGO
2	0,0230064	3768	CARTAGENA
3	0,024076	7985	ALGECIRAS
4	0,025434	9886	LAS_PALMAS
5	0,025825	1272	VALENCIA
6	0,026406	5432	BARCELONA
7	0,02455	172	BILBAO
8	0,023355	1465	GIJON
9	0,025889	2273	TENERIFE
10	0,025849	968	ALMERIA
11	0,024228	968	ALMERIA
12	0,023783	2273	TENERIFE
13	0,023221	1465	GIJON
14	0,023274	172	BILBAO
15	0,024143	5432	BARCELONA
16	0,024144	1272	VALENCIA
17	0,024449	9886	LAS_PALMAS
18	0,02563	7985	ALGECIRAS
19	0,02503	3768	CARTAGENA
20	0,021369	4405	VIGO
21	0,022405	968	ALMERIA
22	0,02283	1465	GIJON
23	0,021798	172	BILBAO
24	0,021965	2273	TENERIFE
25	0,024703	5432	BARCELONA
26	0,022354	1272	VALENCIA
27	0,026382	9886	LAS_PALMAS
28	0,023953	7985	ALGECIRAS
29	0,022123	3768	CARTAGENA
30	0,021588	4405	VIGO

Tabla 18 Resultados consultas PORT

LENGTH			
PRUEBAS	TEIMPO DE EJECUCIÓN	RESULTADOS	ESLORA
1	0,254149	10000	30
2	0,216635	10000	40
3	0,201028	10000	50
4	0,191552	10000	60
5	0,183668	10000	70
6	0,17252	10000	80
7	0,154896	10000	90
8	0,149217	10000	100
9	0,145587	10000	110
10	0,133815	10000	120
11	0,130584	10000	130
12	0,118918	10000	140
13	0,117291	10000	150
14	0,103056	10000	160
15	0,101822	10000	170
16	0,09827	10000	180
17	0,079956	10000	190
18	0,071942	10000	200
19	0,06352	10000	210
20	0,05999	10000	220
21	0,050151	10000	230
22	0,044698	10000	240
23	0,043569	10000	250
24	0,04043	10000	260
25	0,039969	10000	270
26	0,034739	10000	280
27	0,034177	10000	290
28	0,028807	10000	300
29	0,025366	10000	310
30	0,025195	10000	320

Tabla 19 Resultados consultas LENGTH

POS			
PRUEBAS	TIEMPO DE EJECUCIÓN	RESULTADOS	MMSI
1	0,031291	168	210401000
2	0,020777	77	245878000
3	0,021837	70	244750895
4	0,025231	110	373796000
5	0,012301	23	205362390
6	0,023189	181	224829000
7	0,026328	147	538006963
8	0,025377	159	256789000
9	0,024602	161	250104640
10	0,024836	142	231112000
11	0,02531	122	219699000
12	0,023343	161	212518000
13	0,022346	147	255806113
14	0,019724	64	265902200
15	0,023076	151	248375000
16	0,025416	163	367324580
17	0,026061	155	219024194
18	0,025812	148	251435110
19	0,022488	141	259465000
20	0,031733	148	276182000
21	0,027564	148	244870674
22	0,012594	25	538007455
23	0,024162	177	538003719
24	0,022928	143	351354000
25	0,013944	27	440122610
26	0,017104	53	440204790
27	0,025543	132	440581000
28	0,026077	152	235068109
29	0,024846	168	636013350
30	0,025504	172	636016726

Tabla 20 Resultados consultas POS

CELL			
PRUEBAS	TIEMPO DE EJECUCIÓN	RESULTADOS	CELDA H3
1	0,175181	4186	89195a
2	0,052457	9207	891870
3	0,110454	10000	891929
4	0,025644	1757	89196d6e1
5	0,027615	1506	891f2501
6	0,026227	3210	890c5ac
7	0,037126	6523	892af621
8	0,032179	4654	8939468
9	0,018043	729	8928de90
10	0,033424	5085	8928f17
11	0,024038	1952	8948954e
12	0,012981	1	8922cc85a
13	0,017274	322	89264051ac
14	0,024373	2711	8922d05
15	0,022399	97	8916542
16	0,012956	9	8936e4c
17	0,019516	1014	8929a19
18	0,014093	10000	8922d0b
19	0,007212	2711	8922d05
20	0,210782	10000	894446
21	0,01613	426	891d2428
22	0,216762	10000	894444
23	0,018766	1724	8906dd
24	0,014233	107	8939b60
25	0,074828	10000	891951
26	0,043484	46	892954
27	0,013225	1	8916cc3b
28	0,15843	10000	89196b
29	0,013342	86	892320
30	0,020081	912	8911359

Tabla 21 Resultados consultas CELL

TIME			
PRUEBAS	TIEMPOS DE EJECUCIÓN	RESULTADOS	MMSI, TIEMPO (MIN)
1	0,021083	4	235097309, 30
2	0,010941	7	232106000, 40
3	0,010176	5	367467980, 60
4	0,01075	9	249338000, 70
5	0,010652	9	440122460, 90
6	0,009408	5	258084000, 100
7	0,009455	15	309891000, 120
8	0,010218	12	431008813, 140
9	0,010178	18	367581980, 160
10	0,010004	19	538007680, 180
11	0,01015	22	367715840, 200
12	0,009836	24	367560590, 220
13	0,010056	26	338125429, 240
14	0,010429	20	367423000, 260
15	0,009677	32	219153000, 280
16	0,009698	35	249209000, 340
17	0,010006	22	538071019, 360
18	0,009556	20	218781000, 400
19	0,009605	50	232004406, 450
20	0,010007	21	244742000, 500
21	0,009787	48	251250110, 550
22	0,009779	61	235005541, 600
23	0,010025	54	232011198, 650
24	0,010417	41	246218754, 700
25	0,009801	42	244740580, 900
26	0,009922	86	538008326, 1000
27	0,01289	23	367123550, 1100
28	0,010488	97	338953000, 1200
29	0,008227	119	366986680, 1300
30	0,008736	124	259635000, 1400

Tabla 22 Resultados consultas TIME

MMSI			
PRUEBAS	TIEMPOS DE EJECUCIÓN	RESULTADOS	MMSI
1	0,019513	163	367324580
2	0,022875	161	257071110
3	0,01955	127	538005691
4	0,01988	97	518132000
5	0,014854	83	563063300
6	0,011549	37	636018704
7	0,024698	163	316005721
8	0,020472	104	255806010
9	0,02379	156	367331760
10	0,02122	162	247387600
11	0,017767	105	224157350
12	0,021082	120	316004106
13	0,022535	157	368004570
14	0,015039	60	258473000
15	0,02274	130	357688000
16	0,02582	181	538004697
17	0,019556	125	219215000
18	0,0206	128	564639000
19	0,01474	66	256927000
20	0,019567	114	209028000
21	0,010797	26	538007758
22	0,020833	144	306258000
23	0,019243	128	375665000
24	0,02466	148	564822000
25	0,018457	105	312081000
26	0,022421	144	657179400
27	0,021853	141	659561000
28	0,008666	12	219009000
29	0,017925	93	213065000
30	0,01142	43	21879000

Tabla 23 Resultados consultas MMSI

NEAR			
PRUEBAS	TIEMPOS DE EJECUCIÓN	RESULTADOS	INSTANTE TEMPORAL
1	0,023663	0	1613278
2	0,017156	2	1613295
3	0,01193	0	1613295987
4	0,012355	0	16132885
5	0,011711	0	1613288567
6	0,011628	0	161328609
7	0,011255	0	1613286091
8	0,012222	0	161328608
9	0,010968	0	1613286087
10	0,011064	0	16132852
11	0,011669	0	16132846
12	0,011606	0	16132845
13	0,010733	0	16132844
14	0,010527	0	16132843
15	0,011233	0	16132840
16	0,010878	0	16132839
17	0,011609	0	16132788
18	0,010934	0	16132825
19	0,010468	0	16132994
20	0,010647	0	16132983
21	0,010704	0	161329836
22	0,010516	0	16132965
23	0,01121	0	16132875
24	0,010729	1	16132860
25	0,010959	0	16132855
26	0,010918	0	16133125
27	0,011531	0	16133456
28	0,011141	0	16133820
29	0,011007	0	16132976
30	0,010693	0	16132345

Tabla 24 Resultados consultas NEAR

ZONE			
PRUEBAS	TIEMPOS DE EJECUCIÓN	RESULTADOS	COORDENADAS
1	0,048252	2432	ESTRECHO
2	0,031216	2432	ESTRECHO
3	0,026983	2432	ESTRECHO
4	0,029636	2432	ESTRECHO
5	0,030723	2432	ESTRECHO
6	0,027598	2432	ESTRECHO
7	0,03214	2432	ESTRECHO
8	0,02864	2432	ESTRECHO
9	0,024606	2432	ESTRECHO
10	0,029682	2432	ESTRECHO
11	0,028598	2432	ESTRECHO
12	0,028076	2432	ESTRECHO
13	0,030549	2432	ESTRECHO
14	0,028526	2432	ESTRECHO
15	0,030848	2432	ESTRECHO
16	0,029966	1382	FINISTERRE
17	0,024849	1382	FINISTERRE
18	0,028637	1382	FINISTERRE
19	0,022354	1382	FINISTERRE
20	0,026563	1382	FINISTERRE
21	0,023491	1382	FINISTERRE
22	0,025877	1382	FINISTERRE
23	0,021203	1382	FINISTERRE
24	0,020379	1382	FINISTERRE
25	0,027568	1382	FINISTERRE
26	0,023133	1382	FINISTERRE
27	0,029173	1382	FINISTERRE
28	0,021244	1382	FINISTERRE
29	0,028481	1382	FINISTERRE
30	0,028481	1382	FINISTERRE

Tabla 25 Resultados consultas ZONE

NEARGEО			
PRUEBAS	TIEMPOS DE EJECUCIÓN	RESULTADOS	MMSI, INSTANTE TEMPORAL, DISTANCIA
1	0,036943	4	219009171, 1613279618000, 10km
2	0,040367	212	310013000, 1613279618000, 10km
3	0,021017	39	244750888, 1613279618000, 5km
4	0,026748	150	2710445707, 1613279618000, 7km
5	0,022771	57	263701950, 1613279618000, 8km
6	0,028073	88	538071155, 1613279618000, 15km
7	0,017626	9	367105510, 1613279618000, 3km
8	0,016404	5	440686000, 1613279618000, 8km
9	0,01765	12	369916000, 1613279618000, 2km
10	0,028057	112	205277890, 1613279618000, 5km
11	0,020535	31	250004944, 1613279618000, 10km
12	0,019072	9	2585526810, 1613279618000, 1km
13	0,027794	68	257558000, 1613279618000, 15 km
14	0,017825	3	431342000, 1613279619000, 7km
15	0,01636	2	367477410, 1613291913000, 10km
16	0,020568	33	368142220, 1613291913000, 5km
17	0,034866	232	249000934, 1613291913000, 20km
18	0,016373	8	376758000, 1613291913000, 5km
19	0,018492	5	25765900, 1613291914000, 8km
20	0,021753	48	235090245, 1613291914000, 13km
21	0,032477	160	205273990, 1613291914000, 10km
22	0,020257	32	538005182, 1613291916000, 5km
23	0,031237	92	367160860, 1613291916000, 8km
24	0,018092	2	367467980, 1613291916000, 4km
25	0,017329	6	227146,700, 1613291916000, 10km
26	0,016706	13	373765000, 1613291918000, 5km
27	0,034117	90	368084470, 1613291918000, 3km
28	0,022292	33	242177100, 1613291918000, 1km
29	0,032026	155	235102532, 1613291919000, 10km
30	0,019168	32	5670542000, 1613291919000, 5km

Tabla 26 Resultados consultas NEARGEО