



**UNIVERSIDAD EUROPEA DE MADRID**

**ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO**

**MÁSTER UNIVERSITARIO EN ANÁLISIS DE GRANDES  
CANTIDADES DE DATOS-MBI / BIG DATA ANALYTICS-MBI**

**Arquitectura Big Data: Pipeline y Monitorización**

**Enrique Benito Casado**

**CURSO 2018-2019**

Arquitectura Big Data: Pipeline y Monitorización

Enrique Benito Casado

---

**TÍTULO:** ARQUITECTURA BIG DATA: PIPELINE Y MONITORIZACION

**AUTOR:** ENRIQUE BENITO CASADO

**TITULACIÓN:** MASTER UNIVERSITARIO EN BIG DATA ANALYTICS

**DIRECTOR DEL PROYECTO:** JESUS CARRETERO

**EDICIÓN:** 2018- 2019

**CONVOCATORIA:** PRIMERA

**FECHA:** SEPTIEMBRE de 2019

## RESUMEN

Con la realización el proyecto, “Arquitectura Big Data: Pipeline y monitorización”, pretendemos construir un sistema capaz de recorrer el ciclo de vida del dato, desde que este se capture de este, pasando por la ingesta de este mismo, haciendo hincapié en cómo se almacenaría (de manera distribuida) pero además queremos poner foco en todo lo que conlleva una monitorización en una infraestructura Big Data, tanto de Logs como de los datos que se han ido produciendo.

La ingesta de datos se hace a través de unos scripts que recorren la web descargándose las noticias de economía. Apache Kafka es utilizado como para el soporte y la distribución de la llegada masiva de eventos. El almacenamiento de datos de manera distribuida corre a cargo de MongoDB una base de datos orientada a documentos donde se nuestros datos se almacenarán en formato Json. Por último Elasticsearch tendrá un papel fundamental en el análisis de Logs. La visualización de los mismos corre a cargo de Kibana.

**Palabras clave:** Web-Crawling, Scrapy, MongoDB, Big Data, Kafka, Elasticsearch, Logstash, Kibana.

## ABSTRACT

With this project, "Big Data Architecture: Pipeline and Monitoring", we intend to build a system capable of going through the life cycle of the data, from the moment it is captured, through the ingestion of the same, emphasizing how it would be stored (in a distributed way) but we also want to focus on everything involved in monitoring a Big Data infrastructure, both Logs and data that have been produced.

The ingestion of data is done through scripts that run through the web downloading economic news. Apache Kafka is used for the support and distribution of the massive arrival of events. The storage of data in a distributed way is in charge of MongoDB a database oriented to documents where our data will be stored in Json format. Finally, Elasticsearch will play a fundamental role in the analysis of Logs. Kibana is in charge of their visualization.

**Key words:** Web-Crawling, Scrappy, MongoDB, Big Data, Kafka, Elasticsearch, Logstash, Kibana.

**AGRADECIMIENTOS**

A mi tutor Jesús Carretero, por ayudarme en los momentos que necesite de su ayuda, y por llevar mi TFM.

A mi hermano Jaime, que siempre me ayuda, es ejemplo a seguir y fuente de admiración constante.

A mi familia. A Frau Gelii.

# Índice

RESUMEN .....	3
ABSTRACT .....	3
Capítulo 1. INTRODUCCIÓN .....	10
1.1 Planteamiento del problema .....	10
1.2 Objetivos del proyecto .....	11
1.3 Plan de trabajo y descripción técnica.....	11
Capítulo 2. IDEA DE NEGOCIO, INVERSIÓN CON BIG DATA y MACHINE LEARNING.....	15
2.1 Introducción .....	15
2.2 Planteamiento del problema .....	15
2.3 Modelo de negocio .....	17
Capítulo 3. CAPTURA DE DATOS, WEB-CRAWLER .....	19
3.1 Introducción a Web-Crawler .....	19
3.1.1 Scrapy .....	19
3.2 Selectores XPath.....	20
3.2.1 Introducción .....	20
3.2.2 Identificando nuestro XPath.....	21
3.3 Construyendo nuestro Spider .....	23
3.3.1 Paginación .....	24
3.4 Evitando ser baneado.....	24
3.4.1 Técnicas usadas por los administradores web para evitar que les hagan Web-Crawler .....	25
3.4.2 Web-Crawling Best Practices.....	25
3.5 Integrar la ingesta a nuestra arquitectura .....	26
Capítulo 4. Distribución llegada masiva de eventos, Apache KAKFA .....	27
4.1 Introducción a Kafka.....	27
4.1.1 Fundamentos de Kafka.....	28
4.1.2 Teoría de Kafka.....	29

4.2	CLI Kafka (Command Line Interface) .....	31
4.2.1	Introduction.....	31
4.2.2	Kafka topics CLI.....	31
4.2.3	Kafka consumer y Producer .....	32
4.2.4	Kafka y Java .....	32
4.3	Kafka en nuestra arquitectura Big Data .....	33
4.3.1	Creando nuestro Producer .....	33
4.3.2	.....	35
	Creando nuestro consumer.....	35
Capítulo 5.	Almacenamiento: MongoDB y Elasticsearch.....	36
5.1	Introducción .....	36
5.1.1	Scale Up vs Scale out .....	36
5.2	MongoDB vs Elasticsearch .....	37
5.3	MongoDB en nuestra Infraestructura .....	39
5.3.1	Schema on read vs Schema on write .....	39
5.3.2	Bases de datos Primaria y tolerancia a fallos .....	40
Capítulo 6.	Análisis de Logs, ELK Stack .....	43
6.1	Introducción .....	43
6.2	Arquitectura de Elasticsearch .....	44
6.2.1	Nodos .....	44
6.2.2	Índices y documentos.....	45
6.3	Elasticsearch API.....	46
6.3.1	API REST – estado .....	46
6.3.2	API REST - Índices y documentos .....	48
6.4	Beats.....	51
6.4.1	Introducción .....	51
6.4.2	Filebeat.....	52
6.4.3	Metricbeat.....	54
6.4.4	Packetbeat.....	55
6.4.5	Auditbeat.....	56
6.4.6	Libbeat.....	56
6.5	Logstash.....	56
6.5.1	Introducción .....	56

6.5.2	Instalación y funciones.....	58
6.5.3	Monitorizando nuestra infraestructura: Filebeat, Logstash, Elasticsearch.....	60
6.5.4	Creado un mapa de coordenadas mediante la geolocalización: Mapping .....	62
Capítulo 7.	Visualización de datos, Kibana .....	65
7.1	Introducción .....	65
7.2	Discover .....	65
7.3	Visualizaciones y Dashboards.....	67
7.4	APM .....	69
Capítulo 8.	Machine Learning - X-PACK.....	70
8.1	Machine Learning sobre los logs del servidor.....	70
8.2	ML aplicado a actividades de usuario. ....	72
Capítulo 9.	CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO .....	73
Capítulo 10.	Referencias.....	74
BIBLIOGRAFÍA.....		75

# Índice de Figuras

Figura 1:Vista completa proyecto .....	14
Figura 2: Idea de negocio .....	18
Figura 3:Prensa salmón .....	19
Figura 4:Author Xpath .....	22
Figura 5:Fecha , xpath .....	22
Figura 6: Seleccionando los tags .....	23
Figura 7:Texto.....	23
Figura 8:De la Web a nuestro Script.....	24
Figura 9:Colección de spiders.....	26
Figura 10: Inserción de datos original / antigua.....	27
Figura 11: Inserción de datos en un sistema Big Data .....	27
Figura 12: Kafka en acción.....	28
Figura 13: Topics en Kafka.....	29
Figura 14: Replicación en Kafka.....	30
Figura 15:Resumen teoría de Kafka .....	31
Figura 16:Creando un topic.....	32
Figura 17:Producer y consumer .....	32
Figura 18: Nuestro consumer escucha lo que le mandamos mediante java .....	33
Figura 19: Kafka en nuestra arquitectura.....	33
Figura 20:Elasticsearch vs MongoDB .....	34
Figura 21:Escalar en vertical vs escalar en horizontal.....	37
Figura 22:Elasticsearch vs MongoDB .....	37
Figura 23:Ambas Bases de datos en combinación .....	39
Figura 24:Data Integración dentro de MongoDB.....	40
Figura 25:Kafka y MongoDB .....	41
Figura 26: Modulo ELK .....	43
Figura 27: Modelo OSI.....	44
Figura 28: Índice vs Tabla .....	45
Figura 29:Comprobación estado del clúster mediante Dev Tools .....	46
Figura 30: Estado del clúster .....	47
Figura 31:Comprobación del estado de nuestro clúster.....	48
Figura 32: Comprobación de Índices.....	48
Figura 33: Insert SQL vs Insert Elasticsearch.....	49
Figura 34: Lectura de datos .....	49
Figura 35:Borrado y consulta en Kibana .....	50
Figura 36: Update en Kibana .....	50
Figura 37: Realización de consultas .....	51
Figura 38: Ejemplo de Log que estamos recogiendo .....	52

Figura 39: Modificando filebeat.yml para poner la ruta de nuestros logs.....	53
Figura 40: Kibana y Filebeat .....	53
Figura 41: Metricsbeat en Kibana .....	54
Figura 42: Packetbeat, secuencia de ejecucion .....	55
Figura 43: Packetbeat secuencia de ejecución .....	55
Figura 44: Tratamiento de eventos en Logstash .....	57
Figura 45: Test y monitorización de Logstash .....	58
Figura 46: Salida de Logstash .....	58
Figura 47: Primera transformación en Logstash .....	59
Figura 48: Transformación en Logstash II .....	59
Figura 49: Activando la Geolocalización.....	60
Figura 50: Modificando el archivo Logstash.yml para hacer nuestra ETL.....	61
Figura 51: La localización está llegando correctamente .....	62
Figura 52:Plantilla personalizada .....	63
Figura 53: Plantilla de Apache .....	63
Figura 54:Logs de entrada de apache, dibujados en un mapa.....	64
Figura 55: ELK-Kibana .....	65
Figura 56: Visualizando nuestros logs en discover l.....	66
Figura 57: Añadiendo un filtro a nuestra visualización .....	66
Figura 58:Barchar cortado por response .....	67
Figura 59: Visualización Donuts .....	68
Figura 60: Visualización tabla .....	68
Figura 61: Uniendo Todo, dashboard final.....	69
Figura 62: Subiendo a elasticsearch nuestra Data .....	70
Figura 63: Aplicando ML a nuestros logs .....	71
Figura 64: Observando de cerca las anomalías.....	71
Figura 65: Aplicando ML a métricas de usuarios .....	72

# Capítulo 1. INTRODUCCIÓN

El trabajo de fin de master está mayoritariamente enfocado desde el punto de vista de un Ingeniero de Datos, es decir la persona que se va a encargar de crear los sistemas y la infraestructura para proveer datos allá donde sean necesarios. Además, este proyecto hace foco en todo lo que corresponde a una monitorización de una infraestructura Big Data.

## 1.1 Planteamiento del problema

### Pipeline

La idea de negocio que da forma a este proyecto de Big Data, es una pequeña empresa que quiere empezar a invertir en bolsa basándose en el análisis de opiniones tanto de internet y redes sociales como de la prensa escrita especializada en economía.

#### Prensa escrita

Desde los diferentes medios de prensa económica (lo que se conoce como prensa salmón), como el economista, el blog del salmón, merca2 etc... Se escribe diariamente artículos sobre la buena situación de ciertas empresas que luego se derrumban en bolsa y viceversa, entre tanta maraña de artículos realmente es difícil saber de quién fiarse. El ciudadano medio que lee estas noticias de economía no tiene realmente casi ninguna garantía de que aquello que está leyendo sobre economía vaya a ser cierto. La única manera que tendría de hacer esto sería seguir a un autor y ver si a lo largo del tiempo han ido acertando en sus predicciones.

#### Redes sociales(Twitter)

No es nada nuevo, de hecho, se ha hablado varias veces sobre el poder que tiene analizar twitter para predecir el mercado de valores.

Nosotros como empresa vemos twitter como otro factor a analizar, otro medio más de donde recopilar datos y poder estudiarlos. Buscamos poder encontrar patrones y correlaciones entre el comportamiento del mercado de valores.

#### Monitorización.

Lo que ocurre dentro de un clúster de Big data no es una caja negra en la que no sabemos que es lo que está pasando, queremos hacer una monitorización de logs sobre lo que está ocurriendo a nuestra infraestructura.

### ¿Qué vamos a monitorizar?

Siguiendo el hilo conductor de una empresa que recoge los datos de blogs los analiza y quiere invertir en bolsa, esta empresa tiene servidores web donde ofrece nuestros servicios, estos servidores generan logs de usuarios (o agentes, o incluso ataques) que están visitando nuestros servicios.

## 1.2 Objetivos del proyecto

El objetivo de este proyecto es constatar e incluso amplificar los conocimientos obtenidos en el master de “Análisis de grandes cantidades de datos”, haciendo especial hincapié en la asignatura de “Sistemas de gestión de datos e infraestructura”.

## 1.3 Plan de trabajo y descripción técnica

El plan de trabajo en este TFM, va a ser el siguiente:

Siguiendo el hilo conductor de una empresa que necesita, extraer datos de la web, almacenarlos y analizarlos vamos a ir viendo las herramientas que nos van a permitir llevar este proceso a cabo.

Todos los capítulos se presentan de la siguiente manera:

Primero se hace una introducción al capítulo, se presenta de forma teórica de la herramienta que vamos a utilizar, poco a poco vamos subiendo de nivel hasta que llega un apartado donde integramos la herramienta en nuestro proyecto. Todos los capítulos quedan abiertos para líneas futuras donde se debe incluir más y mejores funcionalidades para seguir con la filosofía de desarrollo continuo.

Herramientas que se van a tratar en este TFM

- 1) **WEB – Crawler (Scrapy):** Python ofrece la librería más potente hoy en día para extraer datos de las páginas web, veremos cómo funciona este programa, entenderemos la importancia que tiene X-path para la extracción de datos de páginas webs.

- En un primer momento veremos cómo hacer consultas sobre la página web para encontrar los datos que estamos buscando.
- Una vez identifiquemos que campos o datos nos interesan, escribiremos nuestro script capaz de capturarlos de manera automática.
- El último paso será ser capaz de escribir una primera pipeline donde al lanzar nuestro script los datos se vayan almacenando directamente en la base de datos.

- 2) **MongoDB:** Un proyecto de Big Data, no pude llevarse a cabo sin un almacenamiento de datos distribuidos, tolerante a fallos y que se pueda escalar de manera horizontal, MongoDB será nuestra base de datos elegida.
  - Introducción teórica a MongoDB y como la deberíamos escalar en el futuro, la parte práctica de MongoDB viene dada en el apartado 1, en combinación con nuestro Scrapy y Pymongo.
- 3) **Elasticsearch:** Es una base de datos que comparte todas las características anteriormente mencionadas en MongoDB, solo que en esta ocasión hay que decir que está orientada a ser una search-engine es decir muy óptima para mostrar los datos que están ya insertados
  - Veremos cómo trabajar con Elasticsearch y las diferentes opciones que ofrece como la inserción de nuevos índices, datos, visionado borrado etc...
- 4) **Kafka:** Kafka será nuestra herramienta para gestión de eventos, en este apartado haremos una introducción a Kafka, veremos sus componentes más importantes y crearemos nuestros propios "producer" y "consumer" que enviaran los datos a Elasticsearch.
  - Tendremos que crearnos una cuenta de desarrollador en Twitter y pedirle permiso para poder coger sus datos, una vez twitter nos autorice recibiremos 4 claves de seguridad que serán nuestras credenciales en todos nuestros scripts que interactúen con Twitter.
  - Las contraseñas son únicas y bajo ningún concepto se deben dar a terceras personas, en caso de querer reproducir esta parte la persona tendrá que pedir autorización explícita a Twitter y este tendrá que enviarle las claves.
  - Nos crearemos nuestro "producer" y "consumer" en java e indicaremos que información es la que queremos que sea la que se envíe.
- 5) **Beats:** El stack de ELK nos proporciona "Beats" para todo lo que tiene que ver con la monitorización de nuestra infraestructura. Instalaremos los diferentes servicios de Beats y monitorizaremos nuestro equipo.
  - Instalaremos Metricbeats, para la monitorización de nuestro clúster, en nuestro caso lo que se monitorizara es la máquina virtual que está ejerciendo de simulador de clúster.
  - Instalaremos Filebeat para monitorizar logs, como no contamos con un apache instalado que nos haga de servidor web que genera logs, estos los simularemos para ello utilizaremos un programa de github escrito en Python que se llamada "**Fake-Apache-Log-Generator**" y nos generara los logs para que posteriormente los podamos analizar.
  - Instalaremos Auditbeat que monitorizara todo lo relativo a la seguridad dentro de nuestra infraestructura, sirve para decir que usuario ha entrado en determinado directorio que hayamos especificado.

**6) Logstash:** Otra herramienta que nos proporciona ELK que nos permite hacer pequeñas transformaciones en los datos antes de enviarlos a Elasticsearch. En nuestro proyecto haremos diferentes transformaciones en Logstash para limpiar los logs.

- Utilizaremos las diferentes funciones que ofrece Logstash y finalmente crearemos una pequeña ETL que será capaz de leer datos que han sido dejado por el generador de logs falso.
- Crearemos un mapa en Kibana que nos muestre mediante ips de donde están viniendo estos logs.

**7) Kibana:** La última de las herramientas que nos proporciona ELK y que vamos a utilizar en este proyecto. Kibana es una herramienta de monitorización muy poderosa que va unida a Elasticsearch.

- En este apartado crearemos diferentes visualizaciones de nuestro sistema para posteriormente unirlas y crear nuestro propio Dashboard. Los datos utilizados para este apartado son los mismo que teníamos en el apartado 6, los creados mediante el generador de Logs.
- Utilizaremos el módulo de pago de ELK (en nuestro caso probaremos la versión de prueba de 30 días para llevar a cabo este proyecto) X-Pack, su versión más avanzada para hacer aprendizaje automático sobre nuestros logs. Haremos dos casos de uso de ML y veremos cómo detectar comportamientos anómalos sobre nuestros logs.

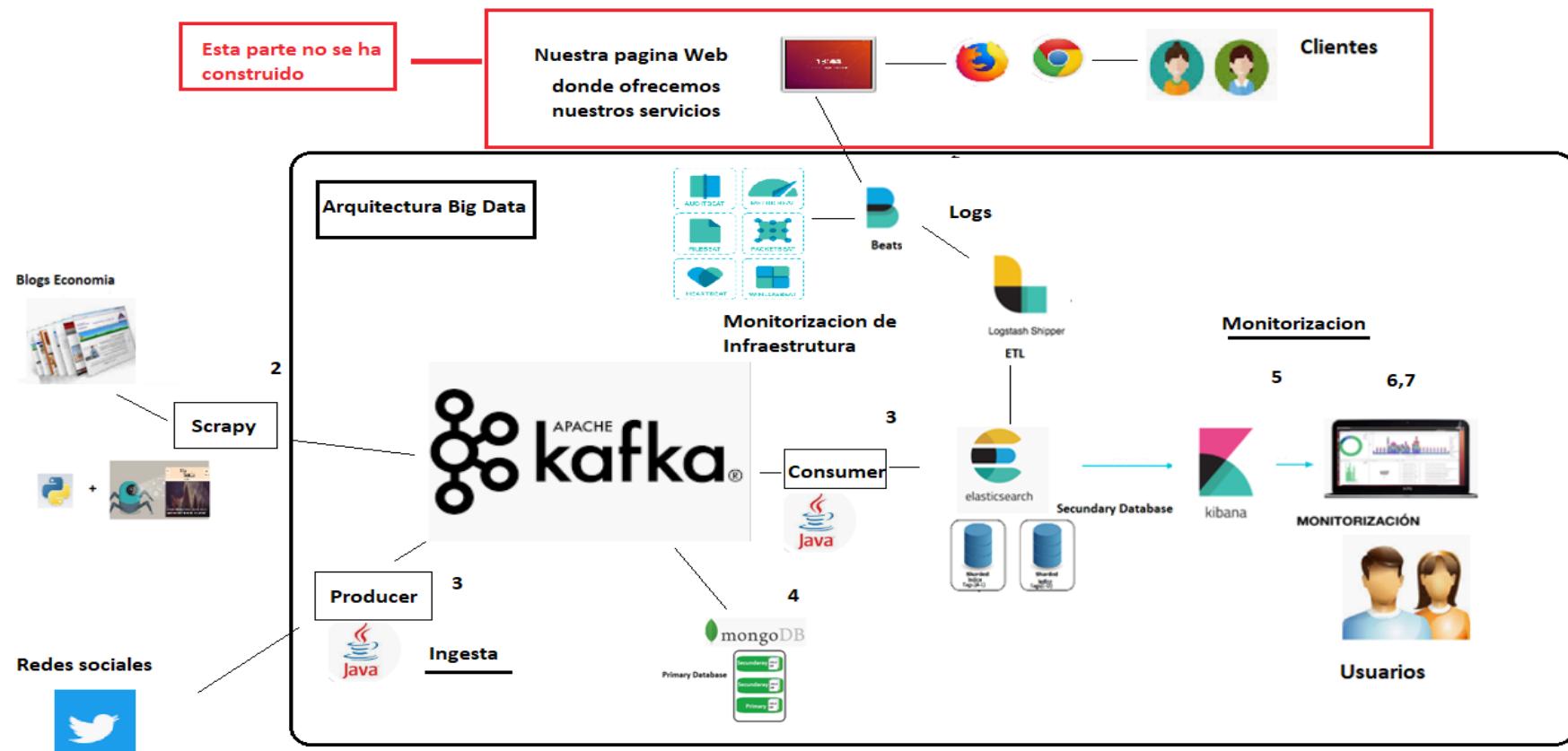


Figura 1: Vista completa proyecto

# Capítulo 2. IDEA DE NEGOCIO, INVERSIÓN CON BIG DATA y MACHINE LEARNING

## 2.1 Introducción

Enriis-Consulting es una empresa que cree en la posibilidad de invertir en bolsa y ganar dinero si somos capaces de identificar la información correcta que está en internet.

Para realizar dichas inversiones va a enfocarse en 2 fuentes de datos

- 1) Opiniones de los escritores de medios de economía.
- 2) Análisis de Twitter y redes sociales.

Una de las frases más repetidas en el mundo del Big Data es “Los datos son el nuevo petróleo” desde Enriis-Consulting realmente lo creemos, los datos (el petróleo) están ahí, necesitamos simplemente encontrarlos, recopilarlos y analizarlos.

## 2.2 Planteamiento del problema

En este apartado queremos poner el foco en que problema hemos venido a solucionar, que aporta nuestra empresa al mundo, que valor aportamos. Desde nuestra filosofía de empresa tenemos claro que no se puede ganar dinero sin aportar valor, es decir si queremos ganar dinero es porque estamos aportando un producto de calidad. A continuación, presentamos los siguientes desafíos que han motivado la creación de esta empresa.

Actualmente cualquier persona puede leer diferentes medios de economía y lanzarse a invertir en función de las opiniones que lea de los diferentes expertos, pero nos plantea los siguientes desafíos.

### Prensa escrita

***¿Qué fiabilidad tienen los llamados expertos financieros que escriben cada día en las secciones de economía?***

Desde los diferentes medios de prensa económica (lo que se conoce como prensa salmón), como el economista, el blog del salmón, merca2 etc... Se escribe diariamente artículos sobre la buena

situación de ciertas empresas que luego se derrumban en bolsa y viceversa, entre tanta maraña de artículos realmente es difícil saber de quién fiarse.

***¿Se puede invertir en mercados financieros guiándonos solo de la opinión de algunos escritores?***

La siguiente pregunta lógica que podríamos hacernos es, ¿se podría llegar a ganar dinero invirtiendo en bolsa siguiendo solo los consejos de estos economistas más confiables?

Hay un dato que nos ha animado a impulsar este proyecto de investigación y no es otro que el famoso experimento “Un paseo aleatorio por Wall Street”

“Para comprobar si los aciertos de los expertos eran o no aleatorios debía hacerse, según Malkiel, un concurso entre profesionales y una elección de acciones completamente al azar. La metáfora de esta selección fortuita consistía en imaginar un mono con los ojos vendados lanzando dardos a la página con la lista de acciones del *The Wall Street Journal*. Luego se compararían los rendimientos de las carteras de ambos contendientes... Lo sorprendente es que cuando se comparó el comportamiento anual de la cartera de valores elegida al azar por el mono, con el de los fondos de inversión referenciados al mercado estadounidense, la cartera del mono había superado al 85% de los fondos”.

El principal problema de este experimento era que trataba a todos los gestores de inversión como un bloque, y esto no era así, habría gente cuyas predicciones siempre eran buenas y otros cuyas predicciones eran malas.

***¿Puede el ciudadano medio que lee noticias de economía fiarse de la opinión de estos expertos?***

El ciudadano medio que lee estas noticias de economía no tiene realmente casi ninguna garantía de que aquello que está leyendo sobre economía vaya a ser cierto, La única manera que tendría de hacer esto sería seguir a un autor y ver si a lo largo del tiempo han ido acertando en sus predicciones.

***¿Puede la inteligencia artificial gracias a la recopilación de miles de datos históricos y la ayuda del NLP (Natural Language Processing) aportar claridad a esta cuestión?***

Cuál sería la consecuencia de juntar cantidades ingentes de Datos con el poder de la Inteligencia artificial. Los datos vendrían a modo de Terabytes de la recopilación de artículos que se han escrito en internet en los apartados de prensa económica.

El poder de NLP sería hacer un análisis de sentimiento, es decir poder responder a la siguiente pregunta: ¿Está el autor hablando positivamente/negativamente/neutro sobre la compañía/producto/valor en el futuro? De manera automática se podría responder a esta pregunta gracias a Machine Learning y su análisis reconocedor de texto.

Una vez hemos recopilado los datos y los hemos procesado para saber si un autor hace una crítica positiva negativa o neutra de un producto falta la tercera pata que sería compararlo con cómo ha evolucionado en el mercado. De tal manera que se podría saber qué porcentaje de acierto tiene un escritor sobre economía. Posterior se puede seguir analizando más en profundidad.

#### **¿Qué porcentaje tiene de acierto sobre un tema en concreto?**

Es posible que nuestro escritor no sea una persona fiable cuando habla de macroeconomía, pero sin embargo cada vez que habla blockchain siempre acierta, en este sentido sería interesante tenerle en cuenta solo cuando habla del tema que realmente entiende.

#### **¿Qué fiabilidad tiene a X meses?**

Es posible que el escritor a corto plazo suele acertar, pero a largo y medio plazo no o viceversa.

#### **Redes Sociales**

En el caso de las redes sociales en nuestro caso Twitter, nos plantea otra serie de desafíos, por un lado podríamos plantearnos las mismas preguntas que en el apartado anterior sobre diferentes personajes que escriben sobre economía a analizar si sus opiniones son relevantes o no, pero además nos abre la posibilidad de análisis masivo de sentimientos, tendencias etc...

Nuestro equipo de científicos de datos será el encargado de encontrar estas correlaciones entre datos y comportamiento del mercado de valores. Además, no es un tema nuevo, sino que se tiene constancia de estudios realizados sobre dicho tema, y nos consta que es posible.

## **2.3 Modelo de negocio**

Desde Enriis-Consulting tenemos dos modelos de negocio.

#### **Invertir nuestro dinero.**

Nosotros realmente creemos en nuestro producto y por tanto somos los primeros que vamos a utilizar nuestro dinero para invertir en bolsa basándonos en las recomendaciones de nuestro sistema somos nosotros. Una vez hemos identificado que autores tienen credibilidad y sobre qué temas contamos un porcentaje de acierto para invertir en los productos de los que está

hablando, nuestra idea es invertir en dichos productos y obtener un retorno económico. Este es realmente el núcleo de nuestro negocio pues si no somos capaces de hacer análisis correctos y ganar dinero de nuestras inversiones no vamos a poder vender nuestra cartera de inversión a los clientes.

### Cartera de inversión para clientes.

Nuestro modelo de negocio pasa también por ofrecer nuestros servicios a los clientes que quieran depositar su confianza con nosotros, de los beneficios de estos clientes obtendríamos un porcentaje. Enriis-Consulting cuenta con una página web de empresa donde ofrecemos nuestros servicios de inversión a clientes particulares.

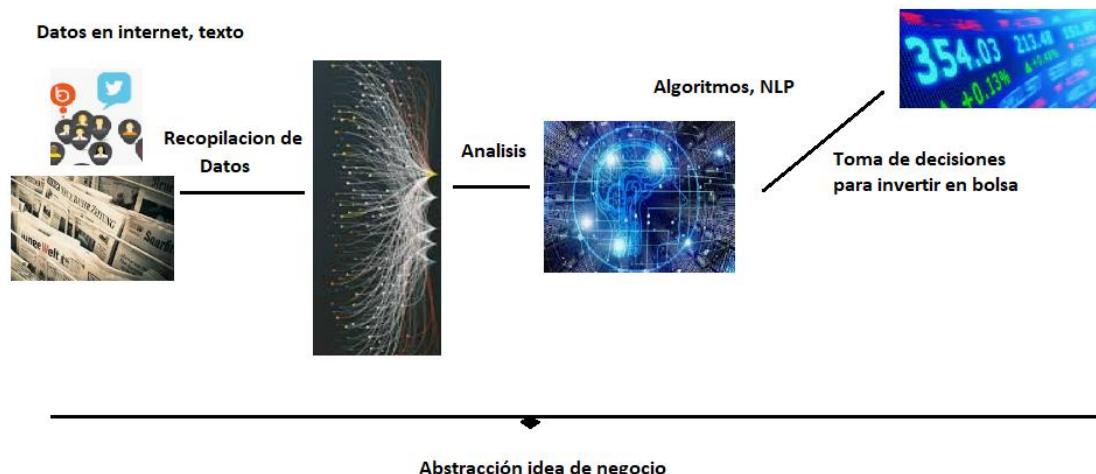


Figura 2: Idea de negocio

## Capítulo 3. CAPTURA DE DATOS, WEB-CRAWLER

Los datos con los cuales nos vamos a nutrir en este proyecto provienen de la prensa económica escrita, más en concreto de la conocida como prensa salmón (merca2, el Economista, Expansión, Cinco Días, El Blog Salmon).



Figura 3:Prensa salmón

### 3.1 Introducción a Web-Crawler

Web-Crawler es un término que se utiliza para referirnos al concepto de extraer los datos de las páginas web. Todo lo que se ve en una página web puede ser objeto extraído, esto incluye, texto, imágenes, videos, emails etc...

#### 3.1.1 Scrapy

Scrapy es un framework *open source*, para la recolección de datos de las páginas web. Anteriormente existían diferentes librerías capaces también de hacer Web-crawling como por ejemplo BeautifulSoup, el problema es que estas librerías no eran aptas para proyectos realmente complejos. Un factor diferencial de Scrapy sobre otros frameworks es la paginación, Scrapy permite de una manera sencilla extraer la información no solo de una página web sino de navegar por páginas anteriores de la web que estamos visitando. Esto es un elemento fundamental pues lo que tratamos es de descargarnos el histórico.

El framework de Scrapy se basa en 5 componentes que se describen a continuación.

##### 3.1.1.1 Spiders

En el componente Spider es donde definiremos que es lo que queremos extraer de la página web en cuestión. Existen 5 tipos diferentes de Spiders (ScrapySpider, CrawlSpider, XMLFeedSpider, CSVFeedSpider, SitemapSpider).

### **3.1.1.2 Pipelines**

Este componente se utilizará, para hacer limpieza de datos, remover los duplicados y guardar datos en una base de datos externa.

### **3.1.1.3 Middlewares**

Este componente tiene todo lo necesario en relación a la petición y respuesta (Request/Response) a una página web.

### **3.1.1.4 Engine**

Es el encargado de la coordinación de todos los componentes anteriormente mencionados.

### **3.1.1.5 Scheduler**

Es el responsable de garantizar el orden en las operaciones.

## **3.2 Selectores XPath**

### **3.2.1 Introducción**

Antes de construir nuestro script en Scrapy tenemos que saber qué es lo que realmente nos interesa de la página web en cuestión, es decir no queremos descargarnos todo el contenido de la página web pues esta está llena de metadatos que no nos interesan para nuestro análisis futuro.

Cada sitio web que queramos hacer “crawling” necesitaremos tener un script específico adaptado a tu estructura XML.

XPath(XML Path Lenguaje) es un “Query lenguaje” usado para seleccionar nodos de documentos XML o HTML. Existen 4 tipos de nodos:

- **Element Node:** Representa los “tags” en un documento HTML  
`<p></p>, <h1></h1>, <div></div> ...`
- **Attribute Node:** Representan atributos de un Element Node  
`@href, @id, @class.`
- **Comment Node:** Representa el comentario que hace en un documento HTML.
- **Text Node:** Representa el texto que se contiene dentro de un Element Node.

### 3.2.2 Identificando nuestro XPath.

Lo primero que tenemos que tener claro es saber que datos nos van a interesar para nuestro análisis, es decir que atributos vamos a querer insertar en nuestra base de datos, para su posterior análisis. Nuestro análisis identifica cuatro pilares fundamentales: Autor, Tags, Texto, Fecha de publicación.

Dentro de nuestra prensa de economía vamos a describir como se construye nuestro Xpath para “el Blog del Salmon”:

<https://www.elblogsalmon.com/economia/estos-indicadores-apuntan-a-una-burbuja-en-bitcoin-y-otras-criptomonedas>

Para encontrar con las herramientas de desarrollo de google Chrome se tienen que seguir los siguientes pasos:

- 1) Elegir elemento que nos interesa.
- 2) Botón derecho, inspeccionar, esto nos abrirá las ventanas de herramientas de desarrollo.
- 3) Ctrl + F para encontrar buscar.

Lo que necesitamos ahora es identificar la instrucción que nos lleve a los atributos que necesitamos.

#### Autor

Lo primero y más importante será recoger, el nombre del autor que ha escrito el artículo en el periódico, necesitamos este campo para que cuando analicemos los comentarios que ha escrito y lo comparemos como se ha comportado el mercado en los meses futuros podamos determinar si es una persona de la cual nos debamos fiar de sus opiniones en el futuro.

La instrucción en Xpath que nos lleva a identificar el autor de manera única, es la siguiente:

`//a[@class = "abstract-author"]`

## Arquitectura Big Data: Pipeline y Monitorización

Enrique Benito Casado

**El libre comercio, un éxito incluso para quienes se oponen: el CETA dispara las exportaciones españolas**

### ENTORNO

Pese a la oposición de Podemos y abstención del PSOE, la aprobación del CETA (tratado comercial entre la UE y Canadá) ha aumentado las exportaciones españolas a Canadá, sobre todo del sector automovilístico, y reducido el déficit.

 0 **ERIK** - HACE UN DÍA



Figura 4:Author Xpath

## Fecha de publicación

Necesitamos saber la fecha de publicación pues será el referente en el tiempo en el que el equipo de científicos de datos se basará para saber cómo evolucionó el activo del que habla, respecto a la opinión del autor.

La instrucción en Xpath que nos lleva a identificar la fecha de publicación de manera única, es la siguiente:

```
//time[@class = "article-date"]/text()[1]
```

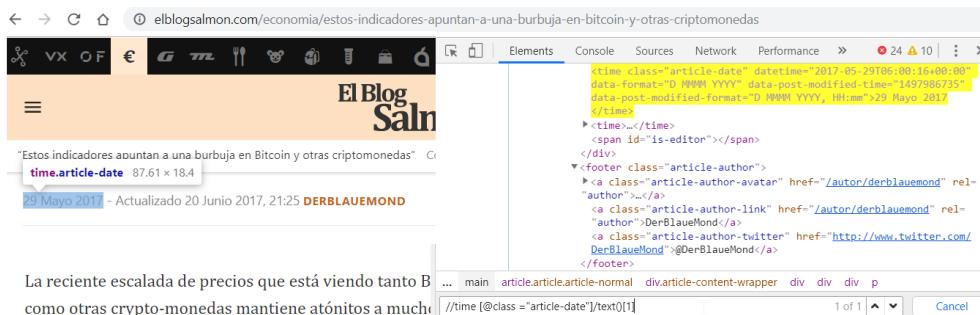


Figura 5:Fecha , xpath

## Tags

Cuando vayamos a procesar el texto se necesitará saber de qué activo financiero se estaba hablando.

```
//a[@class = "article-topic-link"]/text()[1]
```

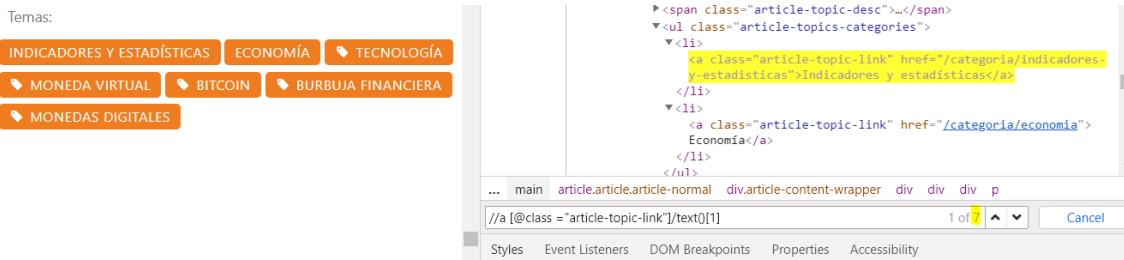


Figura 6: Seleccionando los tags

## Texto

El texto, es el componente principal que se utilizará en el futuro para el análisis. Sobre el texto se efectuará un análisis de sentimiento para saber si el autor estaba vertiendo una opinión positiva, negativa o neutra sobre dicho activo financiero.

Efectuando el análisis de sentimiento sobre el texto, y viendo la evolución del activo financiero en el tiempo se puede determinar si sus previsiones sobre el activo eran correctas o no y por tanto darle más o menos credibilidad al autor del artículo.

```
//div[@class="blob js-post-images-container"]/p/text()[1]
```



Figura 7:Texto

## 3.3 Construyendo nuestro Spider

Una vez sabemos los atributos que necesitamos y los hemos identificado mediante Xpath, lo siguiente que necesitamos es construir un script donde le vamos a pasar dichos atributos que hemos identificado anteriormente.

Los siguientes scripts en Scrapy se tienen que modificar para adaptarlos a el sitio web del que vamos a extraer la información.

*Ítems.py* : Aquí es donde vamos a poner las modificaciones necesarias para que nuestro output en json se vea de la mejor manera posible Removeremos espacios en blanco sobrantes y demás valores que no aporten nada.

*Settings.py*: Aquí indicaremos las instrucciones generales. Es decir si queremos que obedezca al Robot.txt etc..

*Middlewares.py*: Es un framework que se utiliza para procesar las solicitudes y los elementos que se generan a partir de los spiders.



Figura 8: De la Web a nuestro Script

### 3.3.1 Paginación

Una de las razones por la que nos hemos decidido utilizar Scrapy y no otra librería de web-crawler como BeautifulSoup es que Scrapy nos permite fácilmente navegar por la página web, es decir no estamos interesados solamente en descargarnos lo que vemos, como en este caso nos interesa recopilar todo el histórico, debemos ir avanzando por la página web.

En este caso llamaremos a la función:

```
next_page=response.selector.xpath("//a[@class='next_page']/@href").extract first()
```

### 3.4 Evitando ser baneado

Es posible que cuando intentemos descargarnos el contenido de una página web seamos baneados. Este tipo de baneo viene mediante un error: " 429 Too Many Requests HTTP status code"

### 3.4.1 Técnicas usadas por los administradores web para evitar que les hagan Web-Crawler

A continuación, pasamos a detallar las diferentes técnicas que se llevan a cabo.

**Rate Limit Request:** Limita el número de peticiones que una website puede manejar en una cantidad de tiempo, las peticiones tienen que venir de una ip address única. Si se supera el número de peticiones que el administrador del sistema ha establecido como Rate limit, el servidor nos devolverá una respuesta de “http status code 429”.

**User-Agent indefinido:** Una técnica que se utiliza para evitar ser escaneado por los robots es denegar las peticiones a la web si el User-Agent no está definido.

**Detección a través de los “honeypots”:** Se trata de atraer posibles atacantes o bots para ver cómo se comportan.

### 3.4.2 Web-Crawling Best Practices

Una vez que hemos visto como las páginas Webs nos pueden bloquear, explicaremos ahora como evitar ser baneado.

#### 3.4.2.1 Evitar peticiones agresivas

No dejar a nuestro Crawler pedir muchas peticiones de manera muy agresiva. En este sentido un bot de scrapy puede ser considerado como un ataque de DDOS.

#### 3.4.2.2 Ajustar el número de peticiones

Una manera de bajar el número de peticiones es activando en “settings” el atributo “DOWNLOAD\_DELAY” = 5, esto indicaría por ejemplo que haga una petición cada 5 segundos.

#### 3.4.2.3 Activar la extensión “throttle”

Esto lo que hace es que de una manera automática nos ajusta la velocidad de peticiones.

#### 3.4.2.4 Obedecer a ROBOTSTXT\_OBEY

Deberemos activar a “true” el ROBOTSTXT\_OBEY

### 3.4.2.5 Crear un email para User\_Agent

Proveer un email en el USER\_Agent puede hacer que, si las empresas están teniendo problemas con nuestro spider, se decidan a contactarnos.

Todas estas técnicas funcionan para evitar ser baneado de manera temporal, si fuéramos vaneados definitivamente, deberíamos hacer un “Reboot” del router para obtener otra IP Address.

Por último recordar que todas técnicas podrían no funcionar, puesto que hay ciertas páginas webs que utilizan técnicas avanzadas para evitar que se recopilen datos en ellas.

## 3.5 Integrar la ingesta a nuestra arquitectura

A lo largo del capítulo 2 hemos visto como es crear un spider para, hacer Web-Crawler de una página web. Ahora bien, una arquitectura de Big Data evidentemente no se construye si solo necesitáramos extraer datos de una página web, son cientos de páginas webs/blogs/periódicos de los que queremos extraer información.



Figura 9:Colección de spiders

Evidentemente no podemos dejar que cientos de eventos interactúen con la base de datos de manera simultánea, lo que necesitaremos es un gestor de eventos, y es aquí donde entra en juego nuestro siguiente elemento en nuestra arquitectura: KAFKA..

# Capítulo 4. Distribución llegada masiva de eventos, Apache KAKFA

## 4.1 Introducción a Kafka

Para Introducirnos a Kafka, tenemos que remontarnos al capítulo 1, a la parte de cómo hemos insertado los datos en MongoDB. Esa parte ha funcionado correctamente porque estábamos en los inicios, teníamos un script que insertaba en una base de datos, es decir teníamos un sistema de origen y un sistema objetivo.



Figura 10: Inserción de datos original / antigua

El problema es que estamos construyendo una arquitectura Big Data donde no vamos a insertar datos desde un solo sistema sino desde cientos.

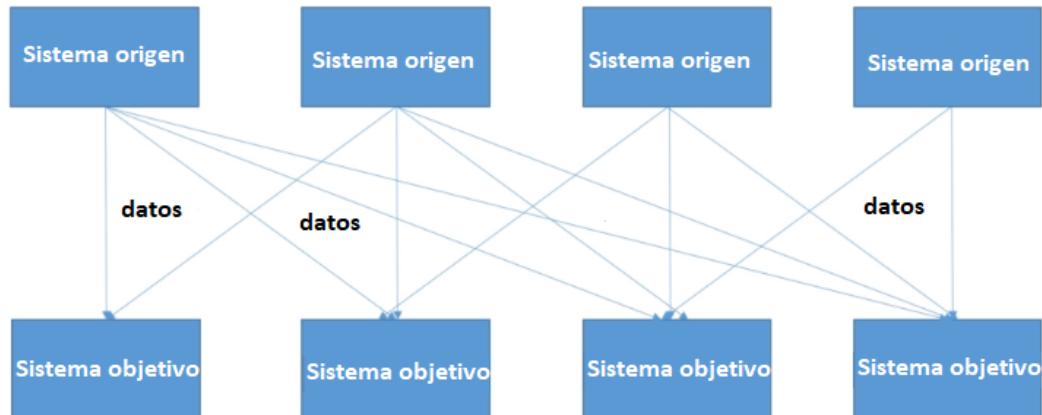


Figura 11: Inserción de datos en un sistema Big Data

Si solo tenemos 4 sistemas objetivo y origen tendríamos que escribir 16 integraciones, en un sistema Big Data ni no van a ser 4 sino muchas más.

Cada integración consiste en:

- Elegir el protocolo: Es decir cómo se transportan los datos (TCP,HTTP,REST,FTP,JDBC..)
- Formato de los datos ( Binary,CSV,JSON,Avro.. )
- Esquema de los Datos.

¿Cómo solucionamos esto?

Aquí es donde aparece Apache Kafka



Figura 12: Kafka en acción

Los datos se distribuyen en Kafka que actúa como mecanismo de transporte, es un sistema de colas. Es usado por más de 2000 compañías, y por el 35% de las 500 empresas más poderosas.

Kafka, es distribuido, y tolerante a fallos.

#### **4.1.1 Fundamentos de Kafka**

Kafka se escala de manera horizontal

Tiene un gran funcionamiento (una latencia de menos de 10ms) – tiempo real.

## 4.1.2 Teoría de Kafka

**Topics:** Es la base de todo en Kafka, representa a una porción de datos, es parecido a una tabla en SQL y se identifica por un nombre.

**Particiones:** Los “topics” se dividen en particiones, cada partición se encuentra de manera ordenada y cada mensaje dentro de una partición obtiene un id incremental llamado offset.



Figura 13: Topics en Kafka

El orden está garantizado dentro de una partición.

Los datos en Kafka solo se mantienen por un tiempo determinado, normalmente una semana. Una vez que se escriben los datos estos no pueden ser cambiados, son inmutables.

**Clúster en Kafka:** Un clúster en Kafka está compuesto por muchos Brokers, un bróker no es más que un servidor. Cada bróker esta identificado por un ID que tiene que ser un entero (no podemos ponerle el nombre que queramos).

**Broker:** Cada bróker no contiene toda la información, sino que una parte de ella, puesto que Kafka es un sistema distribuido. El número mínimo de brokers, que se recomienda es 3 cuando queremos crear un clúster en Kafka.

**Factor de replicación en Kafka:** Podemos elegir el factor de replicación de nuestras partición, obviamente si queremos que nuestro sistema sea tolerante a fallos tiene que ser mayor de 1.

El número que se aconseja, es tener replicación 3.

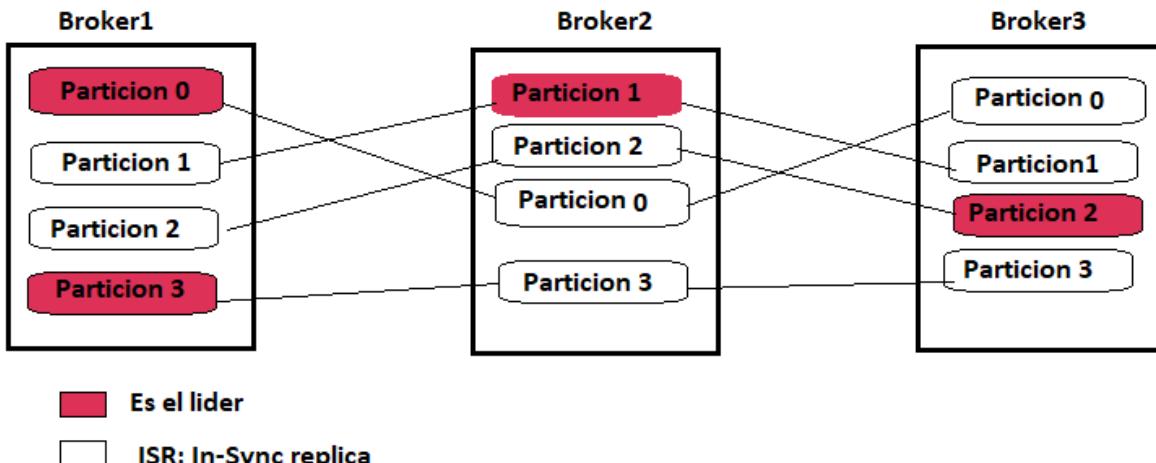


Figura 14: Replicación en Kafka

**Producers:** Escriben los datos a los topics, actúa como un balanceador de carga automático.

**Consumer:** Leen los datos de los topics, los datos son leídos en orden, dentro de cada partición.

**Consumer offset:** Es el mecanismo que tiene Kafka en caso de que un servidor se caiga, para identificar donde debe volver a empezar a leer los topics (para evitar leerlos desde el principio).

**Zookeeper:** Organiza los brokers, ayuda al leader a la elección de las particiones, envía notificaciones a Kafka en caso de cambios (caída de bróker, nuevo topic, borrada de topic etc...). Kafka no puede funcionar sin Zookeeper, y es por eso que lo tenemos que tener arrancado y funcionando antes de poder

Garantías en Kafka:

- Los mensajes se acumulan en las particiones en el orden que han sido enviados.
- Los consumidores leen los mensajes en el orden que han sido guardados (FIFO)
- Con un factor de replicación de N, los productores y consumidores pueden tolerar un fallo de los brokers de N-1.
- Se recomienda tener como factor de replicación 3.

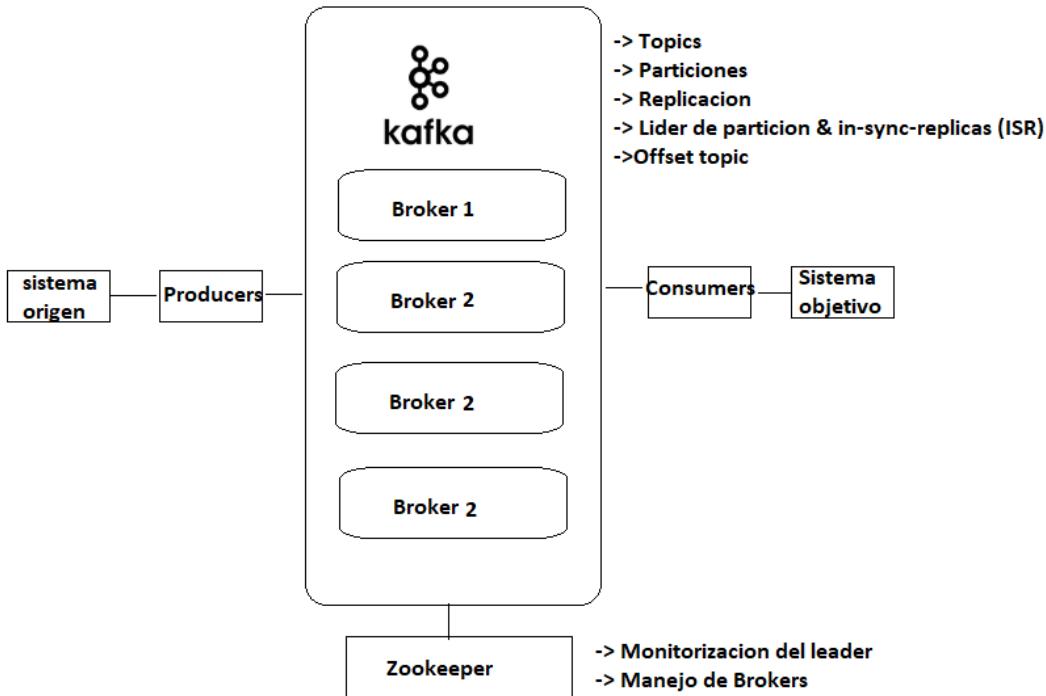


Figura 15:Resumen teoría de Kafka

## 4.2 CLI Kafka (Command Line Interface)

### 4.2.1 Introduction

CLI es la interfaz de comandos que utilizamos para darle órdenes a Kafka. A continuación, vamos a ver las diferentes instrucciones que se puede ejecutar sobre los distintos elementos que tiene.

### 4.2.2 Kafka topics CLI

#### Crear.

Podemos crear un topic manualmente, esto sería como nuestro `create table` de las bases de datos relacionales.

Código: `Kafka-topics.sh -zookeeper 127.0.0.1:2181 -topic blog_01 -create—partitions 3 -replication-factor 1`

```
elastic04@elastic04:~/Desktop/Kafka/kafka_2.12-2.0.0$ kafka-topics.sh --zookeeper 127.0.0.1:2181 --topic blog_01 --create --partitions 3 --replication-factor 2
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore('_') could collide. To avoid issues it is best to use either, but not both.
Error while executing topic command : Replication factor: 2 larger than available brokers: 1.
[2019-09-21 12:40:03,057] ERROR org.apache.kafka.common.errors.InvalidReplicationFactorException: Replication factor: 2 larger than available brokers: 1.
(kafka.admin.TopicCommand$)
```

Figura 16:Creando un topic

### Listar

Podemos ver todos topics disponibles: **Kafka-topics.sh -zookeeper 127.0.0.1:2181 -List**

Efectivamente nos muestra el topic que hemos creado manualmente

### Mostrar

Para ver en detalle un topic: **Kafka-topics.sh -zookeeper 127.0.0.1:2181 -topic blog\_01 -describe**

### Borrar

**Kafka-topics.sh -zookeeper 127.0.0.1:2181 -topic blog\_02 -d**

## 4.2.3 Kafka consumer y Producer

El Producer va a ser el encargado de enviar los datos a Kafka y el consumer de leerlos.

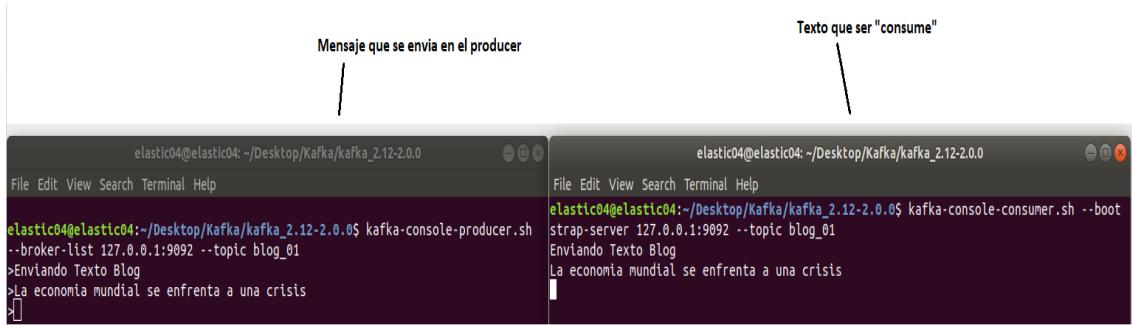
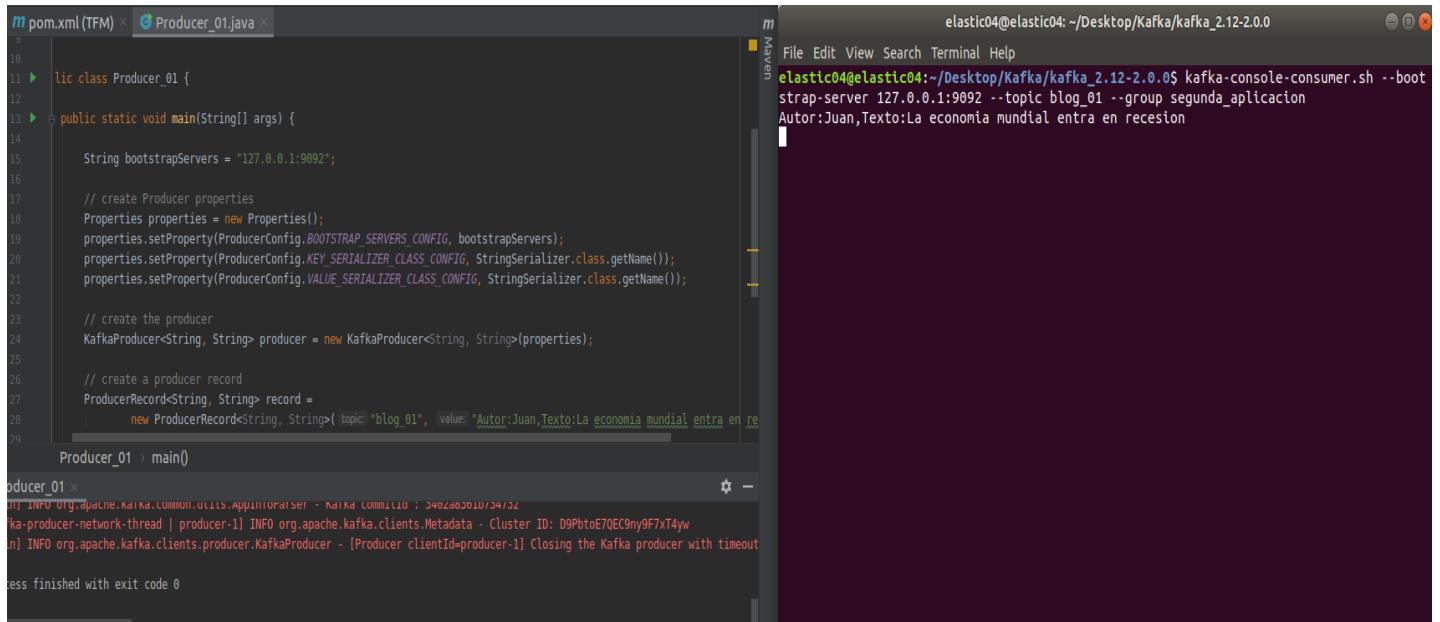


Figura 17:Producer y consumer

## 4.2.4 Kafka y Java

Hemos visto como introducir manualmente mediante comandos nuestros consumers y producers, el siguiente paso es encapsularlo en un programa en Java para que nos de el mensaje. En nuestro archivo pom.xml vamos a poner todas las dependencias que nuestro proyecto va a necesitar.



The screenshot shows a terminal window and an IDE (IntelliJ IDEA) side-by-side. The terminal window on the right displays the command `kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic blog\_01 --group segunda\_aplicacion` and its output, which includes the message "Autor:Juan,Texto:La economía mundial entra en recesión". The IDE window on the left shows the code for a Kafka producer named `Producer\_01.java`.

```

m pom.xml (TFM) <  Producer_01.java <
10
11  lic class Producer_01 {
12
13  public static void main(String[] args) {
14
15      String bootstrapServers = "127.0.0.1:9092";
16
17      // create Producer properties
18      Properties properties = new Properties();
19      properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
20      properties.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
21      properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
22
23      // create the producer
24      KafkaProducer<String, String> producer = new KafkaProducer<String, String>(properties);
25
26      // create a producer record
27      ProducerRecord<String, String> record =
28          new ProducerRecord<String, String>( topic: "blog_01", value: "Autor:Juan,Texto:La economía mundial entra en re
29
Producer_01 > main()

```

Output from the terminal:

```

producer_01.x
[INFO] org.apache.kafka.common.Uuids.AppInfoParser - Kafka COMMITTER : 340265010/34/34
[INFO] Kafka-producer-network-thread | producer-1] INFO org.apache.kafka.clients.Metadata - Cluster ID: D9Pbt0E7QEC9ny9FTxT4yw
[INFO] [n] INFO org.apache.kafka.clients.producer.KafkaProducer - [Producer clientId=producer-1] Closing the Kafka producer with timeout
cess finished with exit code 0

```

Figura 18: Nuestro consumer escucha lo que le mandamos mediante java

## 4.3 Kafka en nuestra arquitectura Big Data



Figura 19: Kafka en nuestra arquitectura

Lo primero que tenemos que hacer es crearnos una cuenta de desarrollador en twitter. Twitter es muy celoso de quien está “escuchando” sus Tweets, puesto que es información muy valiosa.

Para poder trabajar con los tweets hemos tenido que enviar una solicitud a Twitter contando explicando que en este caso se trata de un proyecto de investigación para la universidad. Una vez hemos explicado para que queremos los datos recibimos la aprobación de Twitter. En nuestro clúster de Big Data, cuando pasemos a producción es decir se pretenda tener.

### 4.3.1 Creando nuestro Producer

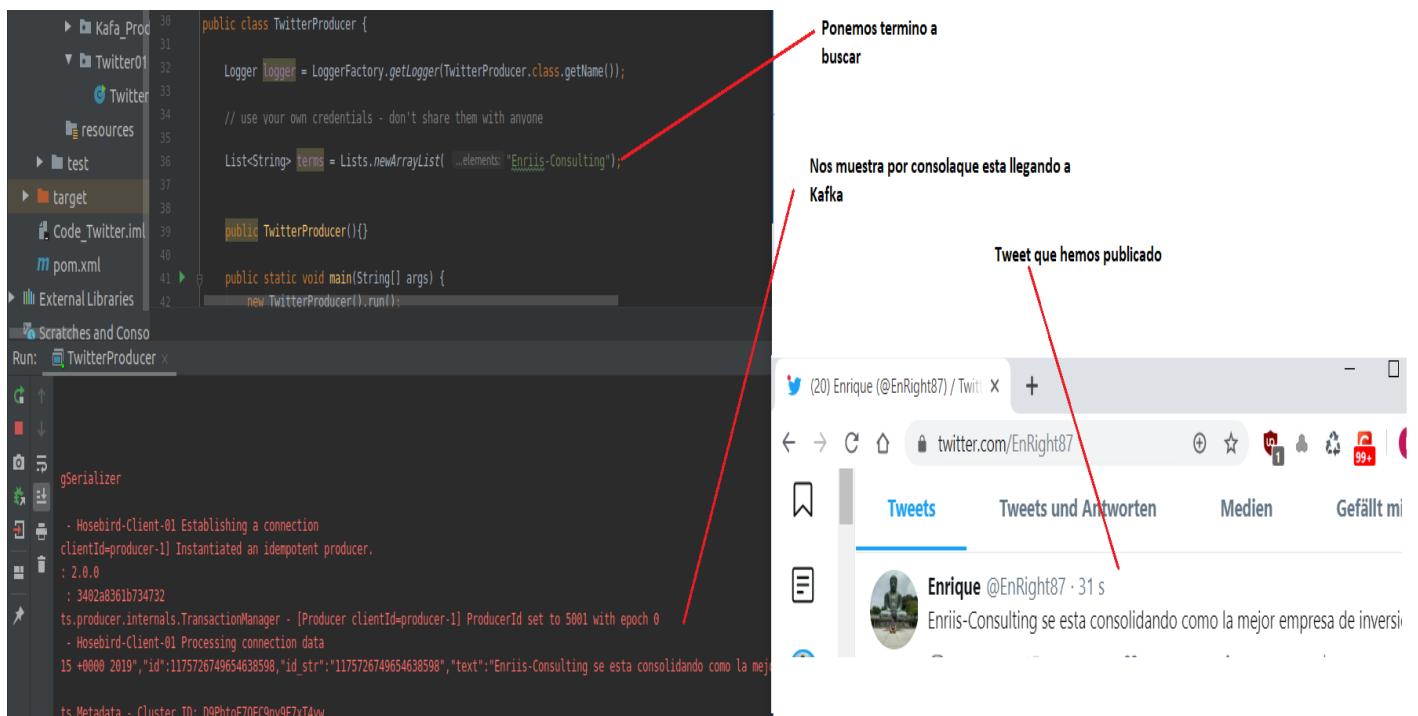
Existen diferentes factores que tenemos que tener en cuenta para construir nuestro producer. Puesto que va a ser una aplicación Java, tenemos que pasarle al archivo pom.xml las dependencias que vamos a necesitar en maven.

Necesitamos:

- Dependencias de nuestro programa en Maven.
- Contraseñas de twitter
- Tener instalado ntp y funcionando. (sin esto nos daba error)

Una vez hemos creado nuestro cliente, creamos el producer, la conexión etc..

Llega el momento de ver si ha funcionado. Para ello añadimos en la lista de palabras a buscar 2 palabras “raras” para que así las podamos visualizar más fácilmente. En nuestro caso para una primera verificación de que funciona ponemos el nombre de nuestra empresa “Enriis-Consulting”.



The image shows a Java IDE interface with two main panes. On the left, the code for a 'TwitterProducer' class is displayed:

```

30 public class TwitterProducer {
31
32     Logger logger = LoggerFactory.getLogger(TwitterProducer.class.getName());
33
34     // use your own credentials - don't share them with anyone
35
36     List<String> terms = Lists.newArrayList( ...elements: "Enriis-Consulting" );
37
38     public TwitterProducer(){}
39
40     public static void main(String[] args) {
41         new TwitterProducer().run();
42     }

```

A red arrow points from the line 'terms = Lists.newArrayList( ...elements: "Enriis-Consulting" );' to the text 'Ponemos término a buscar' (We enter the search term).

On the right, a Twitter profile for 'Enrique (@EnRight87)' is shown, displaying a single tweet:

(20) Enrique (@EnRight87) / Twitter +  
[twitter.com/EnRight87](https://twitter.com/EnRight87)  
Tweets Tweets und Antworten Medien Gefällt mir

Enrique @EnRight87 · 31 s  
 Enriis-Consulting se está consolidando como la mejor empresa de inversi...  
15 +0000 2019, "id":117526749654638598, "id\_str":"117526749654638598", "text":"Enriis-Consulting se está consolidando como la mejor empresa de inversi...

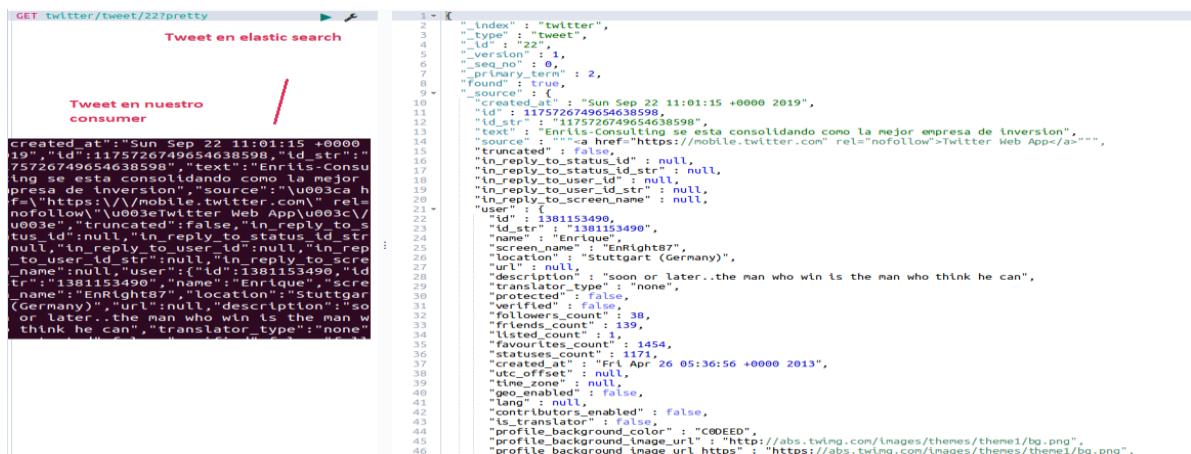
A red arrow points from the tweet text to the text 'Tweet que hemos publicado' (Published tweet).

Figura 20:Elasticsearch vs MongoDB

Una vez que hemos comprobado que funciona, volvemos a poner en la lista de palabras a buscar, las palabras que nos interesa insertar en nuestra arquitectura, que serán palabras que tienen relación con la economía “bitcoin”, “blockchain”, “crisis” etc..

#### 4.3.2 Creando nuestro consumer

Lo siguiente que tenemos que crear es un consumer, que es donde le vamos a decir a Kafka donde queremos que nos lleve los datos. El consumer básicamente coge la información que le esta guardada en Kafka y la envía a sistema objetivo en este caso es nuestra base de datos de Elasticsearch.



```

GET twitter/tweet/22?pretty
Tweet en elastic search
Tweet en nuestro consumer

{
  "_index": "twitter",
  "_type": "tweet",
  "_id": "22",
  "_version": 1,
  "_seq_no": 0,
  "_score": 2,
  "found": true,
  "source": {
    "created_at": "Sun Sep 22 11:01:15 +0000 2019",
    "id": 1175726749654638598,
    "id_str": "1175726749654638598",
    "text": "Enriis-Consulting se esta consolidando como la mejor presa de inversion",
    "source": "https://mobile.twitter.com/rels_nofollow",
    "truncated": false,
    "in_reply_to_status_id": null,
    "in_reply_to_status_id_str": null,
    "in_reply_to_user_id": null,
    "in_reply_to_user_id_str": null,
    "in_reply_to_screen_name": null,
    "user": {
      "id": 1381153490,
      "id_str": "1381153490",
      "name": "Enrique",
      "screen_name": "EnRight87",
      "location": "Stuttgart (Germany)",
      "url": null,
      "description": "soon or later..the man who win is the man who think he can",
      "translator_type": "none",
      "protected": false,
      "verified": false,
      "followers_count": 38,
      "friends_count": 139,
      "listed_count": 1,
      "favourites_count": 1454,
      "statuses_count": 1451,
      "created_at": "Fri Apr 26 05:36:56 +0000 2013",
      "utc_offset": null,
      "time_zone": null,
      "geo_enabled": false,
      "lang": null,
      "country_code_enabled": false,
      "is_translator": false,
      "profile_background_color": "C00EED",
      "profile_background_image_url": "http://abs.twimg.com/images/themes/theme1/bg.png",
      "profile_background_image_url_https": "https://abs.twimg.com/images/themes/theme1/bg.png"
    }
  }
}

```

Figura 17 :Consumer a elasticsearch.

# Capítulo 5. Almacenamiento: MongoDB y Elasticsearch

## 5.1 Introducción

Me gustaría empezar este capítulo con una frase que ha marcado lo que es la filosofía del Big Data

*In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers.*  
— Grace Hopper

Evidentemente, nuestro almacenamiento será distribuido. Uno de los desafíos que nos encontramos en las infraestructuras Big Data es que tienen que estar preparadas para afrontar las vs de Velocidad y Volumen (además de otras), es decir nuestro volumen de datos va a ir creciendo a una velocidad muy deprisa y nuestro volumen va a ser muy elevado.

### 5.1.1 Scale Up vs Scale out

Escalar de manera horizontal no es solo una cuestión de precios, sino de capacidad, escalar de manera vertical solo se podrá hacer hasta el tamaño máximo que soporte una máquina.

#### Procesamiento masivo de datos:

Al tratarse de una infraestructura Big Data con un crecimiento exponencial, descartamos un sistema de bases de datos tradicional, un RDBMS donde el crecimiento es “scaling up” y optaremos por un sistema de bases de datos distribuido, un sistema NoSQL donde podremos escalar horizontalmente “scaling out” de esta manera si necesitamos más poder de almacenamiento bastara con añadir “commodity software” horizontalmente, siendo un proceso mucho más barato y fácil de escalar.

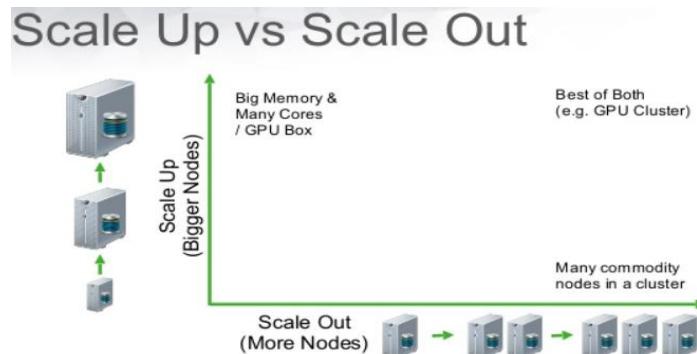


Figura 21: Escalar en vertical vs escalar en horizontal

## 5.2 MongoDB vs Elasticsearch

Una vez nos hemos decidido por un almacenamiento distribuido y fácilmente escalable, llega el momento de elegir la Base de datos vamos a utilizar.

Sabemos que vamos a estar insertando documentos JSON pues es este el formato en el que están trabajando nuestros spiders que recopilan los datos de las páginas Webs. En un principio pensamos en utilizar MongoDB y Elasticsearch puesto que cumplen los requisitos fundamentales y ambas tienen muchas cosas en común, son bases de datos NoSQL, son distribuidas, tiene replicación en shards, ambas tienen un escalado horizontal lo que las hace perfectas para grandes cantidades de datos y trabajan con Documentos JSON, MongoDB para los documentos y una base datos Elasticsearch para monitorizar la infraestructura los logs etc...



Figura 22: Elasticsearch vs MongoDB

Sin embargo en medio de la realización de este TFM, surgieron las siguientes dudas.

***¿Porque la necesidad de dos bases de datos Distribuidas si en vez de solo una?***

Si MongoDB y Elasticsearch comparten características de escalado (horizontal) , trabajo con documentos, replica etc... y además Elasticsearch nos permite trabajar con logs y monitorizarlos y lo más importante es más rápido buscando en documentos que MongoDB.

Realmente ¿qué propósito tiene una base de datos ? uno de ellos y de los más importantes es soportar querys del usuario para devolver la información que se está buscando en el, si Elasticsearch hace esta función mucho más rápido..

***¿Porque tener mongodb y no quedarnos solo con elasticsearch?***

Efectivamente existe una razón por la cual una no es competencia de la otra, es decir ambas son productos muy exitosos en el mercado y no compiten por el mismo nicho, Elasticsearch es una search-engine y MongoDB es una Document-base engine hay dos puntos que nos lleva a Dos puntos principalmente responden a esta pregunta.

**ETL**

Efectivamente insertar en Elasticsearch y luego hacer un proceso de ETL es mucho más complicado que hacerlo en MongoDB, es cierto que Elasticsearch proporciona Logstash donde podemos insertar componentes de la ETL (Filtrado, funciones ,etc) pero no es tan completo como pueda ser una ETL hecha en python donde las posibilidades son infinitas.

**Compatibilidad de ambas Bases de datos.**

Una vez hayamos insertado nuestros documentos pasaran por un proceso de ETL donde los limpiaremos y procesaremos, se hará un volcado de MongoDB a Elasticsearch donde se ejercerán funciones de búsqueda y visualización de datos.

Por tanto ambas bases de datos tienen cabida juntas y se complementan unas con otras.

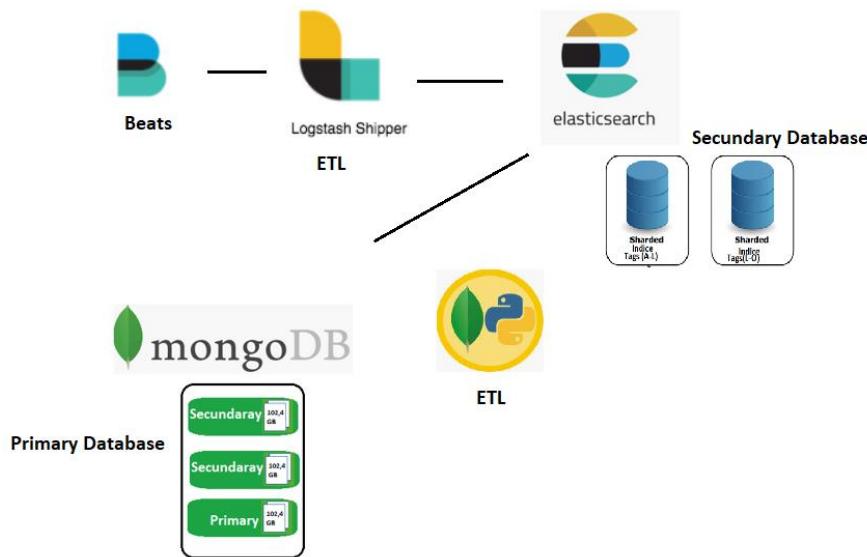


Figura 23: Ambas Bases de datos en combinación

## 5.3 MongoDB en nuestra Infraestructura

MongoDB va a ser nuestra Base de datos primaria, en ella se van a insertar todos los documentos JSON que vayan llegando procedentes de las diferentes fuentes de datos.

### 5.3.1 Schema on read vs Schema on write

El modelo schema on read vs schema on write tiene importancia a la hora de elegir MongoDB como nuestra base de datos primaria, aun habiendo otras alternativas SQL.

**Schema on read:** Significa que para insertar los datos no hace falta que nuestros documentos se correspondan perfectamente con el schema de los documentos previamente existentes, es decir esto es un concepto de almacenamiento, primero almacenamos todos estos Giga so Teras de datos que tenemos ya veremos qué hacemos con ellos, schema on read al contrario de otras bases de datos (sobre todo las Relacionales) solo nos obliga a mantener cierta estructura en los documentos a la hora de leerlos pero no a la hora de almacenarlos, si fuera una base de datos schema on write significaría que si se intentara insertar un documento en un índice determinado pero con otra estructura a la predefinida fallaría. Precisamente antes de leer esos documentos lo que podemos hacer es diferentes procesos de transformaciones sobre ellos en programas como Python gracias a la Api de Mongodb Pymongo, por lo tanto, si tenemos dos fuentes de datos que nos almacenan documentos con estructuras diferentes, primero los almacenaremos luego los trataremos con Python para posteriormente integrarlos y leerlos.



Figura 24:Data Integración dentro de MongoDB

### 5.3.2 Bases de datos Primaria y tolerancia a fallos

#### Tolerancia a fallos y alta disponibilidad.

Optar por un sistema distribuido como solución para nuestro caso de uso, tiene aparte de la ventaja que hemos visto anteriormente de Scaling up vs scaling out, otra ventaja fundamental de que es un sistema tolerante a fallos.

La tolerancia a fallos se trata, fundamentalmente, mediante:

**Replicación:** Consiste en proporcionar múltiples casos idénticos en el mismo sistema o subsistema, dirigiendo las tareas o las solicitudes de todos ellos en paralelo.

**Redundancia:** Consiste en proporcionar múltiples casos idénticos en el mismo sistema y la posibilidad de cambiar a uno de los restantes casos en caso de fallo.

#### Replicación vs Redundancia:

La diferencia entre replicación y redundancia es que, en la redundancia, se duplica todo el sistema cada cierto tiempo mediante un sistema de Backup y en caso de que falle mi sistema principal entonces utilizo mi sistema secundario, en la replicación se duplican también los nodos y estos nodos se están sincronizando entre ellos dentro del clúster.

En nuestro caso particular utilizaremos la técnica de la “replicación” para tener un sistema tolerante a fallos.

MongoDB despliega s en un clúster con múltiples servidores, dos conceptos son clave: **Sharding y Replica**

**Sharding:** Consiste en la partición de los datos entre los servidores que tenemos, los shards se pueden añadir o quitar sin necesidad de llevar la base de datos a offline.

**Replica:** Mantiene a una disponibilidad alta de nuestra base de datos, mediante la copia redundante de la data a través de los servidores. Consta de dos tipos de nodos.

**Primario:** Recibe todas las operaciones de escritura.

**Secundario:** Los nodos secundarios replican las operaciones lógicas del nodo primario a su data set de la manera que la data set primaria queda reflejado en el nodo secundario, si el nodo primario se cae, un nodo secundario elegido pasa a ser el nodo primario.

La configuración mínima que se recomienda de réplica en MongoDB es 3, y consiste en un nodo primario y dos secundarios.

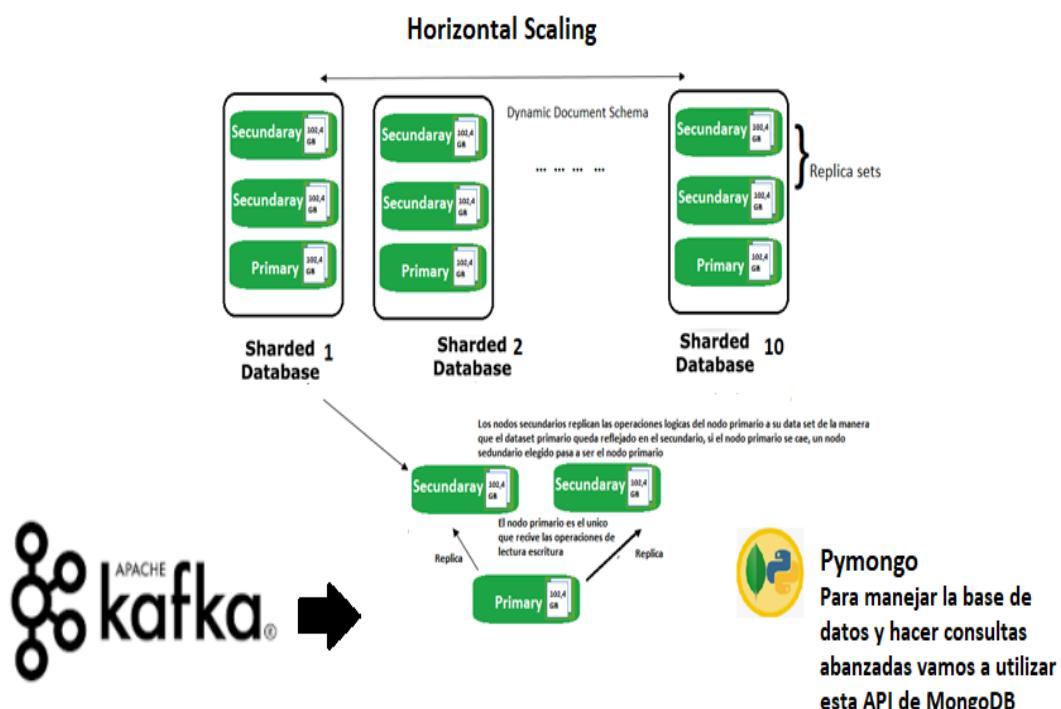


Figura 25:Kafka y MongoDB

### Consideraciones de Hardware y escalado

**Determina cuanta capacidad de datos vamos a tener que afrontar:** Debemos decidir el tamaño del “working set” a través de analizar las “Queries” para saber cuánta cantidad de data vamos a necesitar acceder de una vez, calcular el número de peticiones por segundo que vamos a querer soportar.

- **Hacer una prueba de concepto(POC):** MongoDB te permite hacer una prueba de la aplicación con un 10% del software y del hardware, gracias a esto puedes mejorar el “performance” y corregir fallos
- **Testearlo con un workload real:** No hacer un despliegue hasta testearlo con datos del mundo real.
- **Monitorear y hacer ajustes:** El incremento de usuarios requiere inevitablemente incremento de nuestra capacidad de almacenamiento, nuevos índices etc.

**Nuestra infraestructura Big Data:** La data que se calcula que se genere de extraer información de blogs primer año será de 1TB, puesto que tenemos replica 3, tenemos que afrontar la capacidad de almacenamiento de 3TB. Elegimos un clúster de 10 shards con 3 máquinas por shard de 102,4GB cada una. Los servidores serán 10 Maquinas Linux con una potencia de computo en red de 10 Gigabit.

# Capítulo 6. Análisis de Logs, ELK Stack

## 6.1 Introducción

En una arquitectura Big Data, se producen millones de eventos, cada dispositivo que nuestra empresa tiene conectado, cada servicio que tenemos funcionando genera logs.

### Caso de uso en nuestro proyecto

Nuestra arquitectura Big Data, da soporte a una empresa que quiere hacer inversiones basada en análisis de texto de las opiniones de expertos de economía, pero a su vez ofrece servicios de inversión para clientes, por lo que soporta una página web, en esta página web se conectan personas del resto del mundo, cada vez que alguien se conecta a nuestra página web para ver nuestros servicios, Apache emite un Log cuyo análisis sin las herramientas específicas puede ser complicado. ¿Cómo acceder a la información que necesitamos? ¿Cómo buscamos un valor determinado? En un clúster normal tendríamos que utilizar comandos de Linux como **grep** o **find** para buscar valores, sin embargo, a medida que el número de logs crece se hace muy pesadas estas consultas por no decir casi imposible obtener la información que necesitamos. Pero esto no es todo, en una infraestructura Big Data existen numerosos casos de uso de monitorización y logs: Acceso a determinados directorios, entrada y salida de paquetes de red, seguridad etc...

Al tratarse de cantidades tan grandes de logs es imposible tratarlos todos de manera individual, al no poder analizar todos los logs que se generan no nos es posible extraer información útil. Para ayudarnos **ELK** emerge como la solución a los problemas anteriormente mencionados.

### ELK al rescate

**ELK:** Es el acrónimo de Elasticsearch(Base de datos), Logstash(procesamiento) y Kibana (visualización)

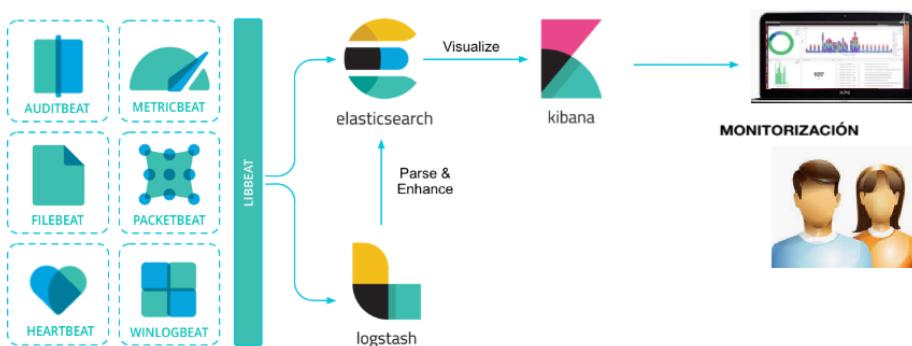


Figura 26: Modulo ELK

## 6.2 Arquitectura de Elasticsearch

Si utilizamos el modelo OSI para ver dónde va a funcionar ElastiSearch, utilizaría la capa 7 y la 4.

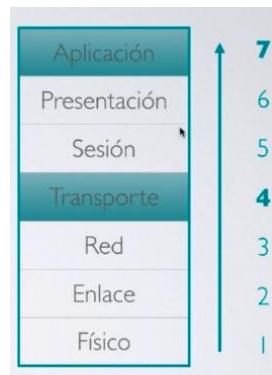


Figura 27: Modelo OSI

### 6.2.1 Nodos

Un nodo simplemente es un equipo o instancia donde se levante un servicio de Elasticsearch, el conjunto de nodos conectados es lo que llamaremos un clúster.

Existen diferentes tipos de nodos.

**Master Node:** Es el nodo que controlara que todo funcione correctamente a lo largo del clúster. El resto de nodos se apoyan en el para llevar a cabo sus acciones. Entre sus tareas están crear índices, indicar que nodos están disponibles y en que nodo se debe almacenar determinado dato. Será necesario que sea lo más estable posibles, y una buena práctica sería que fuera un Nodo dedicado.

**Data Node:** Como su propio nombre indica, será el encargado de almacenar los datos. Realizará operaciones de búsqueda y recuperación de datos, así como operaciones CRUD y crear agregaciones sobre los mismos. Conlleva un gran consumo de recursos. Si se quisiera alcanzar mejores rendimientos se recomendará hacer una escalada de los Data node de manera horizontal.

**Nodos de Ingesta:** Ejecutan tareas de preparación y procesamiento de los datos antes de indexar la información.

**Nodos Coordinadores:** Son básicamente nodos平衡adores de carga.

## 6.2.2 Índices y documentos

Pasamos ahora a ver como se estructura la información existente en los “data nodos”.

**Documentos:** Cada evento que quiera almacenarse en Elasticsearch la denominaremos documento. Los documentos serán almacenados en formato JSON. Además de los datos que los documentos incluyen existen otro tipo de datos no visibles que son los metadatos. Estos metadatos ayudan a Elasticsearch a la gestión.

**Índice:** Un índice no es más que una colección de documentos con características similares. Para que nos hagamos una idea, sería como una tabla en una base de datos estructurada.

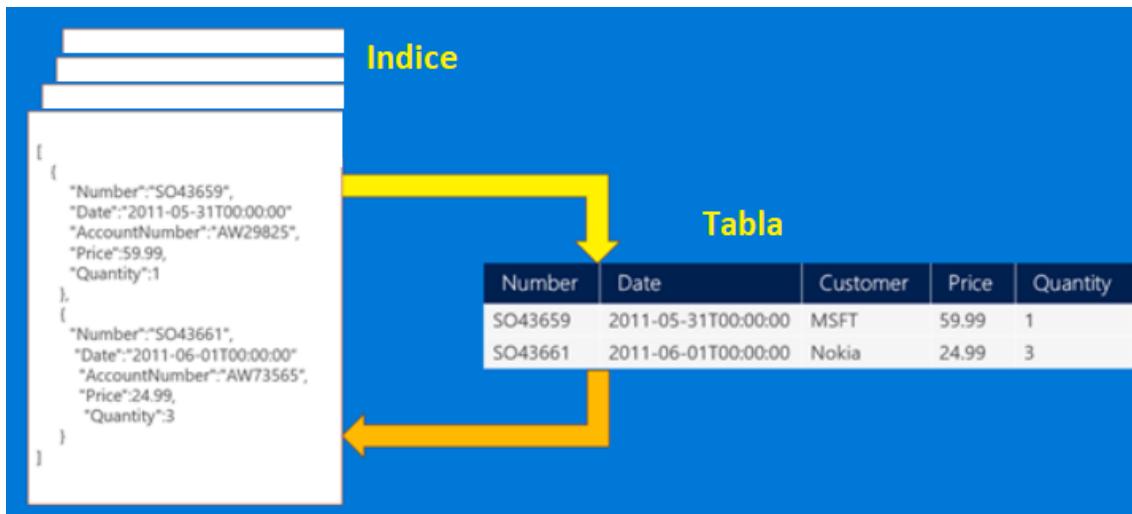


Figura 28: Índice vs Tabla

### 6.2.2.1 Shards y réplicas

**Shards:** La manera que tiene ElasticSearch de almacenar su data de manera distribuida es a través de shards ( al igual que hemos visto en MongoDB).

**Replica:** No es más que una copia exacta de un Shard, con ella obtenemos tolerancia a fallos, la réplica no se encontrara en el mismo nodo.

Aparte de proporcionar tolerancia a fallos, la réplica mejora las consultas ya que si la CPU está trabajando sobre un nodo específico en otra consulta, Elasticsearch puede decidir hacer su consulta en otro nodo que tenga la réplica.

## 6.3 Elasticsearch API

Como ya sabemos las API (Application Programming Interface) son capas de abstracción que ofrecen funciones y procedimientos para comunicarse con aplicaciones. En Elasticsearch existen numerosos tipos de APIs.

### 6.3.1 API REST – estado

#### 6.3.1.1 Cluster API

##### *Cluster Health*

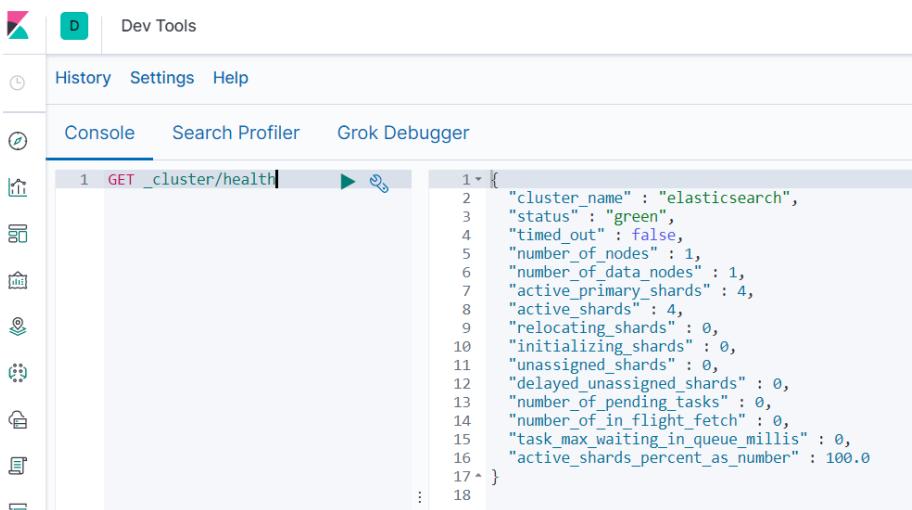
Nos permite consultar el estado del Cluster (nombre,shards,tareas,estado,numero de nodos).Aunque en nuestro caso hayamos montado un único nodo, ese nodo ya pertenece a un cluster.

El parámetro más importante que nos devuelve esta API es “Estado del cluster”:

-**Verde**: Todos los shards han sido indexados con éxito

-**Amarillo**: Los shards primarios se han insertado correctamente, pero con fallo en las replicas

-**Rojo**: Shards primarios no indexados.



```

1 GET _cluster/health
2 {
3     "cluster_name" : "elasticsearch",
4     "status" : "green",
5     "timed_out" : false,
6     "number_of_nodes" : 1,
7     "number_of_data_nodes" : 1,
8     "active_primary_shards" : 4,
9     "active_shards" : 4,
10    "relocating_shards" : 0,
11    "initializing_shards" : 0,
12    "unassigned_shards" : 0,
13    "delayed_unassigned_shards" : 0,
14    "number_of_pending_tasks" : 0,
15    "number_of_in_flight_fetch" : 0,
16    "task_max_waiting_in_queue_millis" : 0,
17    "active_shards_percent_as_number" : 100.0
18 }

```

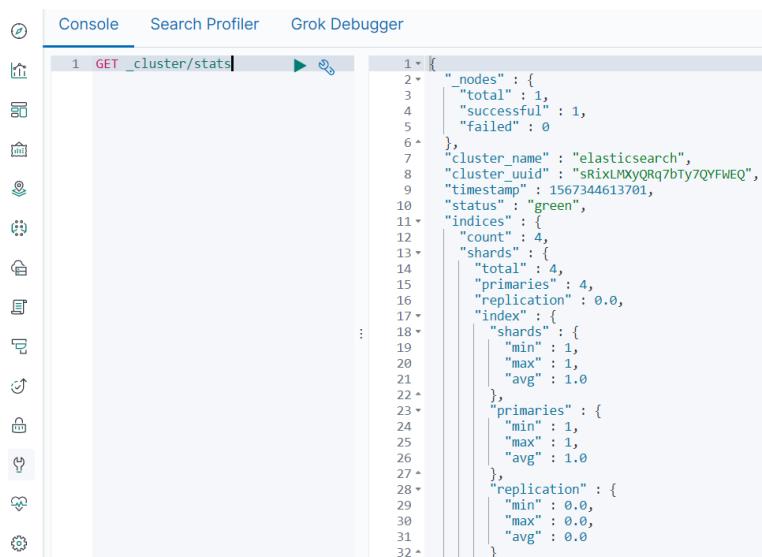
Figura 29:Comprobación estado del clúster mediante Dev Tools

## Nodes Info

Provee información sobre todos los nodos del clúster (puerto HTTP a la escucha, IP, procesos, sistema operativo, nombre del host, plugin..).

## Cluster Stats

Nos permite consultar estadísticas en relación al clúster.



```

1 ~ [{ "nodes" : {
2 ~   "total" : 1,
3 ~   "successful" : 1,
4 ~   "failed" : 0
5 ~ },
6 ~   "cluster_name" : "elasticsearch",
7 ~   "cluster_uid" : "srIXLMxyQRq7btY7QVFWEQ",
8 ~   "timestamp" : 1567344613701,
9 ~   "status" : "green",
10 ~   "indices" : {
11 ~     "count" : 4,
12 ~     "shards" : {
13 ~       "total" : 4,
14 ~       "primaries" : 4,
15 ~       "replication" : 0.0,
16 ~       "index" : {
17 ~         "shards" : {
18 ~           "min" : 1,
19 ~           "max" : 1,
20 ~           "avg" : 1.0
21 ~         },
22 ~         "primaries" : {
23 ~           "min" : 1,
24 ~           "max" : 1,
25 ~           "avg" : 1.0
26 ~         },
27 ~         "replication" : {
28 ~           "min" : 0.0,
29 ~           "max" : 0.0,
30 ~           "avg" : 0.0
31 ~         }
32 ~       }
33 ~     }
34 ~   }
35 ~ }
36 ~ ]

```

Figura 30: Estado del clúster

### 6.3.1.2 Cat API

Es muy similar a la Cluster API pero en este caso, los resultados son mostrados alineados en forma de tabla, lo que los hace mucho más legibles.

Código: [GET \\_cat/health](#)

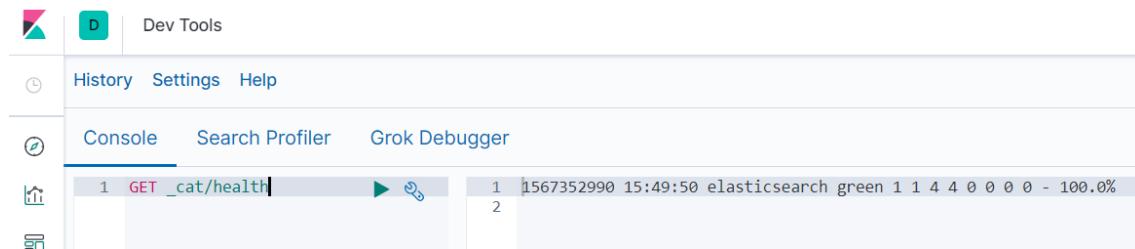


Figura 31:Comprobación del estado de nuestro clúster

### Master Info

Nos proporciona información sobre el nodo máster de un clúster.

Código: GET \_cat/master

### Indices Info

Nos proporciona un listado de índices con su estado (verde, amarillo o rojo), número de shards que lo componen primarios y réplicas, documentos y tamaño en disco.

Código: GET \_cat/Indices

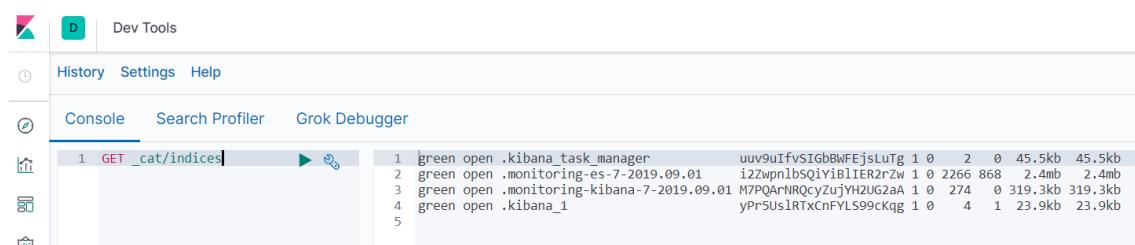


Figura 32: Comprobación de Índices

### Shards Info

Nos Proporciona información sobre el nodo máster del clúster

Código: Get \_cat/shards

## 6.3.2 API REST - Índices y documentos

Enumeramos a continuación las correspondientes APIs para el indexado de los documentos JSON.

### Put

Puede darse el caso que necesitemos insertar un documento manualmente, por la razón que sea, en este caso utilizaremos el comando put y los campos sigan en nuestro índice. Put sería el equivalente a SQL a “**Insert into Tabla values ( value1,...valueN)**”



```

SQL
insert into salmon(Text,Autor,Tag,Fecha) values ("Ha pasado desapercibido en los grandes medios, especialmente comparado con el ruido que generó su puesta en marcha en junio de 2015. Pero tras el cambio de gobierno en Grecia por fin se han levantado los últimos controles de capitales a los que aún estaban sometidos sus ciudadanos.Tras 50 meses en vigor...", "Erlik", "Entorno, Grecia,Banco Central", "30 Agosto 2019")

Kibana
Dev Tools

History Settings Help
Console Search Profiler Grok Debugger

1 POST /salmon/_doc
2 {
3   "Text" : "Ha pasado desapercibido en los grandes medios, especialmente comparado con el ruido que generó su puesta en marcha en junio de 2015. Pero tras el cambio de gobierno en Grecia por fin se han levantado los últimos controles de capitales a los que aún estaban sometidos sus ciudadanos.Tras 50 meses en vigor...",
4   "Autor" : "Erlik",
5   "Tag" : "Entorno, Grecia,Banco Central",
6   "Fecha" : "30 Agosto 2019"
7 }

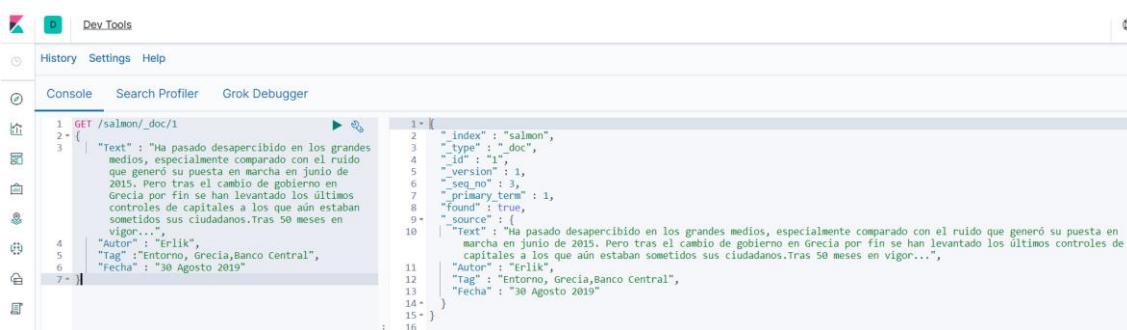
1 {
2   "_index" : "salmon",
3   "_type" : "_doc",
4   "_id" : "83_a7WwBfIq9kibAAAtsd",
5   "_version" : 1,
6   "result" : "created",
7   "shards" : {
8     "total" : 2,
9     "successful" : 1,
10    "failed" : 0
11  },
12  "seq_no" : 2,
13  "_primary_term" : 1
14
15

```

Figura 33: Insert SQL vs Insert Elasticserach

## Get

Permite consultas documentos JSON de un índice a través de su ID. En este caso le damos el id=1



```

Kibana
Dev Tools

History Settings Help
Console Search Profiler Grok Debugger

1 GET /salmon/_doc/1
2 {
3   "Text" : "Ha pasado desapercibido en los grandes medios, especialmente comparado con el ruido que generó su puesta en marcha en junio de 2015. Pero tras el cambio de gobierno en Grecia por fin se han levantado los últimos controles de capitales a los que aún estaban sometidos sus ciudadanos.Tras 50 meses en vigor...",
4   "Autor" : "Erlik",
5   "Tag" : "Entorno, Grecia,Banco Central",
6   "Fecha" : "30 Agosto 2019"
7 }

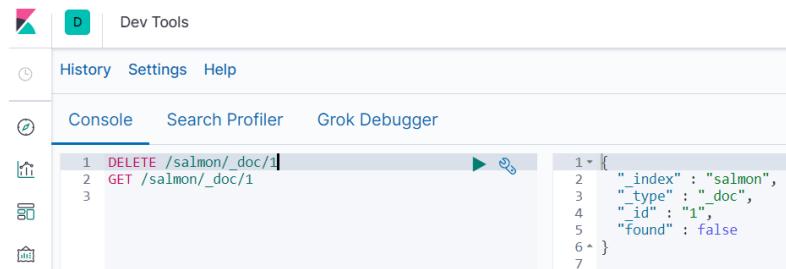
1 {
2   "_index" : "salmon",
3   "_type" : "_doc",
4   "_id" : "1",
5   "_version" : 1,
6   "_seq_no" : 3,
7   "_primary_term" : 1,
8   "found" : true,
9   "source" : {
10     "Text" : "Ha pasado desapercibido en los grandes medios, especialmente comparado con el ruido que generó su puesta en marcha en junio de 2015. Pero tras el cambio de gobierno en Grecia por fin se han levantado los últimos controles de capitales a los que aún estaban sometidos sus ciudadanos.Tras 50 meses en vigor...",
11     "Autor" : "Erlik",
12     "Tag" : "Entorno, Grecia,Banco Central",
13     "Fecha" : "30 Agosto 2019"
14   }
15
16

```

Figura 34: Lectura de datos

## Delete

Permite eliminar documentos de un índice especificando su correspondiente ID. Si intentáramos como en el apartado anterior recoger el Documento, nos daría Falso pues esta ya no existe.



```

1 DELETE /salmon/_doc/1
2 GET /salmon/_doc/1
3
4
5
6
7

```

```

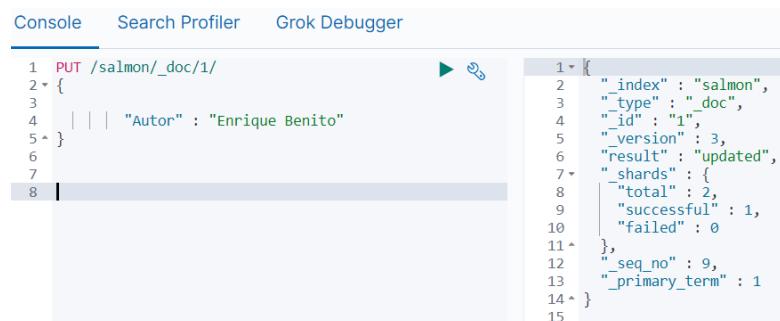
1 {
2   "_index": "salmon",
3   "_type": "_doc",
4   "_id": "1",
5   "found": false
6 }
7

```

Figura 35:Borrado y consulta en Kibana

## Update

Nos Permite actualizar un documento basado en un script. El funcionamiento es que consulta el documento, aplica el script, ósea nuestro update, lo que queramos añadir y vuelve a indexar el documento.



```

1 PUT /salmon/_doc/1/
2 {
3   "Autor": "Enrique Benito"
4 }
5
6
7
8

```

```

1 {
2   "_index": "salmon",
3   "_type": "_doc",
4   "_id": "1",
5   "_version": 3,
6   "result": "updated",
7   "shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12  "seq_no": 9,
13  "_primary_term": 1
14 }
15

```

Figura 36: Update en Kibana

## Search API

URI Search En la propia consulta se pueden especificar los parámetros de búsqueda que devolverán las coincidencias encontradas.

Codigo: [GET salmon/\\_search](#)

Esto nos devolvería todos los documentos que tenemos insertados, si quisieramos hacer una consulta simple por nombre de autor, utilizaremos:

Codigo:[GET salmon/\\_search?q=Autor:Erlik](#)

En este ejemplo nos devolvería todos los nombres.

## Search Templates

Existe la posibilidad de realizar consultas DSL en formato JSON.

Sirve para búsquedas habituales y complejas donde solo habrá que cambiar los parámetros.



```

Console Search Profiler Grok Debugger
1 GET salmon/_search?q=Autor:Erlik
13     "relation": "eq"
14   },
15   "max_score": 1.5580825,
16   "hits": [
17     {
18       "_index": "salmon",
19       "_type": ".doc",
20       "_id": "83.a7aM6BfIq9kibAAAt5d",
21       "_score": 1.5580825,
22       "_source": {
23         "Text": "Ha pasado desapercibido en los grandes medios, especialmente comparado con el ruido que generó su puesta en marcha en junio de 2015. Pero tras el cambio de gobierno en Grecia por fin se han levantado los últimos controles monetarios, los que aún estaban sobre el sistema bancario griego. Los que se vivieron durante el corralito griego es historia. El inicio del corralito impuso límites a 60 euros diarios de dinero que cada ciudadano podía retirar del banco, a la vez que se prohibieron las transferencias bancarias al extranjero. Esta medida generó tales colas en los cajeros que a las pocas semanas fue sustituida por un límite semanal de 420 euros, que al menos permitía sacarlo todo de una vez. En estos años, los controles se han ido relajando gradualmente a medida que la economía griega se estabilizaba. A partir de octubre de 2018 los ciudadanos particulares ya no tenían restricciones a la hora de sacar dinero, pero solo podían transferir un máximo de 4.000 euros cada dos meses fuera de sus cuentas. Las empresas, por su parte, necesitaban autorización del banco central para cualquier operación superior a 100.000 euros. A partir de esta semana, ya no hay límites. ¿Qué ha significado el corralito para la economía griega? ¿Cuáles son las expectativas en esta nueva etapa?", "Autor": "Erlik",
24       "Tags": [
25         "Entorno",
26         "Grecia",
27         "Banco Central Europe"
28       ],
29       "Fecha": "30 Agosto 2019"
30     }
31   }
32 ]
33 ]
34 ]
35 }
36

```

Figura 37: Realización de consultas

## Search Templates

Se utiliza para búsquedas habituales y complejas donde solo habrá que cambiar los parámetros.

## Search Shards

Devuelve los shards sobre los que se ejecutará una búsqueda.

## 6.4 Beats

### 6.4.1 Introducción

Beats son los diferentes componentes que envían la información, haciendo una comparación con el mundo real serían como micrófonos que escuchan en alguna parte de nuestra infraestructura. Los destinos principales de los Beats serán o bien Elasticsearch o Logstash.

Vamos a empezar enviando directamente los eventos a Elasticsearch y posteriormente veremos cómo podemos introducir Logstash para su procesado.

La familia de Beats, cada vez va creciendo más, en un principio se crearon 3 y ahora mismo ya van por 7 oficiales. Vamos a hacer un pequeño repaso pos los Beats que debemos incluir en nuestra infraestructura Big Data.

## 6.4.2 Filebeat

Es el más utilizado, como su propio nombre indica actúa sobre ficheros de texto. En nuestro caso vamos a monitorizar ficheros de logs de un Servidor Apache, de un navegador Web.

Nuestra empresa de inversión, ofrecerá sus servicios a clientes, la actividad que nos llegue desde fuera a nuestra infraestructura será muy importante de monitorizar. Queremos monitorizar desde que país se accede, user Ageant, ip .. etc.

### Fake Logs

En este proyecto todavía no se dispone de la página web de nuestra empresa, por lo tanto, estos logs los simularemos, para ello utilizaremos un proyecto de generación de Fake logs.

Fake logs: <https://github.com/kiritbasu/Fake-Apache-Log-Generator>



Figura 38: Ejemplo de Log que estamos recogiendo

Para generar los logs con la librería de Python será:

Código: **Python apache-fake-log-gen.py -n 10 o LOG** (en el caso de que queramos 10 logs). Por otro lado en la configuración de Filebeat, debemos especificar donde debe buscar los logs.

```

GNU nano 2.9.3                               filebeat.yml

# configuration file.

#===== Filebeat inputs =====

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

- type: log

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /home/elastic04/Downloads/Fake-Apache-Log-Generator/*.log
    #- c:\programdata\elasticsearch\logs\*

```

Figura 39: Modificando filebeat.yml para poner la ruta de nuestros logs

Una vez le hallamos dicho a filebeat donde debería buscar los logs, deberemos activar en la configuración la opción de enviarlos a nuestra base de datos Elasticsearch y por ultimo podremos monitorizarlos en Kibana.

Comprobamos que nuestro Filebeat está activo y enviando Beats a Kibana para su monitorización.

Código: [Service filebeat start / service filebeat status](#)

Lo siguiente es comprobar que los Logs están llegando correctamente, como podemos ver esta activo y corriendo.

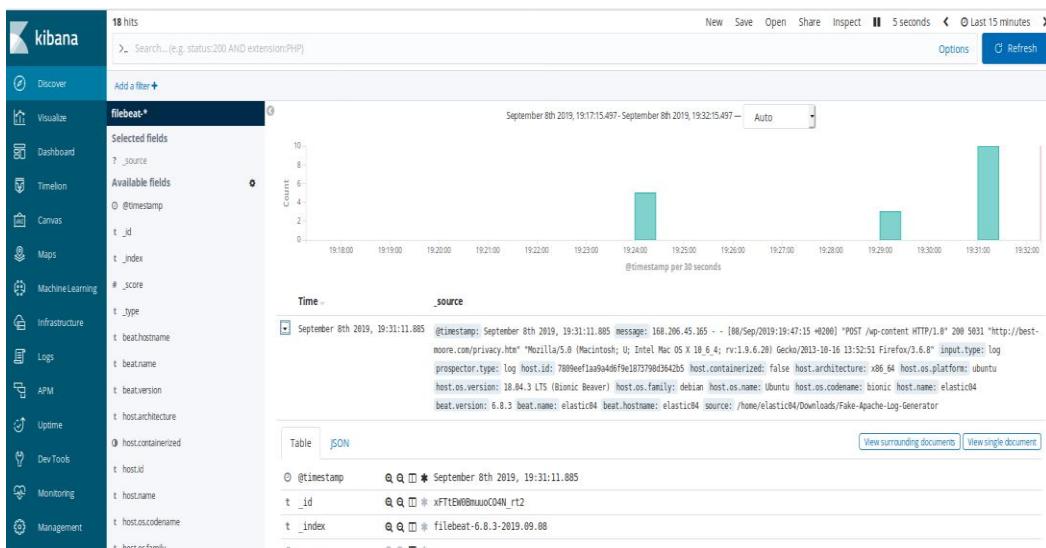


Figura 40: Kibana y Filebeat

El eje de la Y indicara el número de Logs que ha encontrado mientras que en eje X hace referencia a la dimensión tiempo, en nuestra imagen podemos ver que a las 19:24 se han producido 3 logs, a las 19:29 5 y posteriormente a las 19:31 otros 10 logs más.

### 6.4.3 Metricbeat

Metricbeat es un sistema de monitorización de equipo, se encargara de recoger métricas de la infraestructura (CPU,HDD,RAM) y enviarlas. Reporta a nuestra base de datos en Elasticsearch.

Una vez hemos arrancado el servicio, podemos comprobar que está funcionando correctamente con el siguiente comando.

Para ver el estado de metricbeat utilizaremos el comando: **Service metricbeat status**

Para ver los logs utilizaremos el comando: **tail -f /var/log/metricbeat/metricbeat**

La primera vez que queremos activar la carga de visualizaciones entre Kibana y Metricbeat necesitamos activarlo, dicha activación se puede hacer de dos maneras una dentro de Kibana u mediante comando.

**./bin/metricbeat setup --dashboards -c /etc/metricbeat/metricbeat.yml**

Una vez que hemos configurado el envío de logs a Elasticsearch, lo podemos visualizar esta vez ya en la pestaña “Dashboard” de Kibana. El Dashboard se encuentra en System Overview y esta predeterminado, más adelante veremos cómo configurar nuestros propios Dashboards.

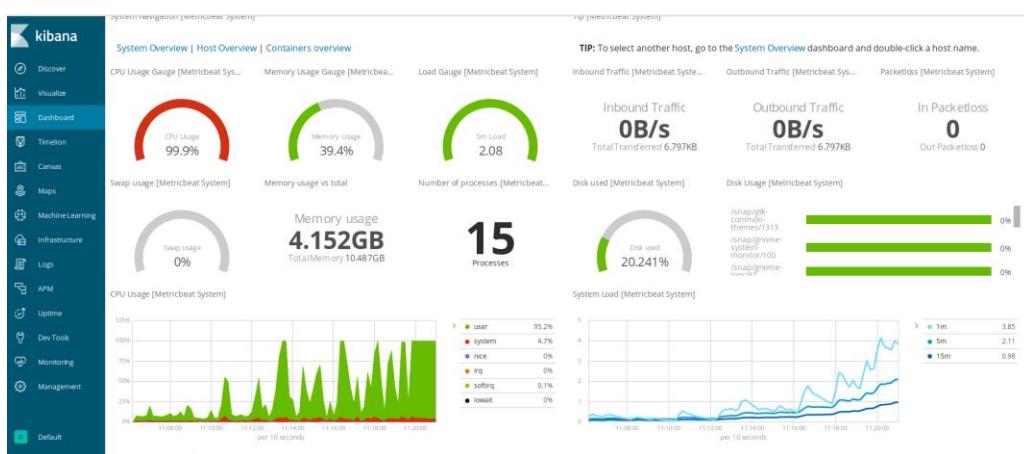


Figura 41: Metricsbeat en Kibana

## 6.4.4 Packetbeat

Es un análisis (sniffer) de paquetes de red (se parece a Wireshark) que reconoce los diferentes protocolos de red, su objetivo es escuchar lo que entra y lo que sale en nuestra infraestructura y mostrarlo de una manera ordenada.

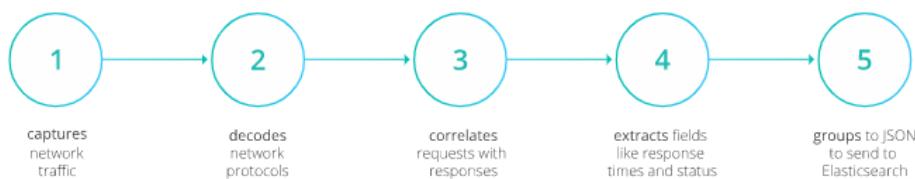


Figura 42: Packetbeat, secuencia de ejecución

Una vez hemos instalado en nuestro equipo packetbeat, comprobamos estatus y lo arrancamos

**Service packetbeat status / Sevice packet beat start**

Una vez que hemos comprobado que se están insertando correctamente como índices de nuestro packetbeat, es tiempo de monitorizarlo y hacer pruebas.

En un principio está vacío nuestro Dashboard lo que deberemos hacer es abrir nuestro navegador y realizar consultas de búsqueda para comprobar como nuestro Dashboard se va llenando de información.

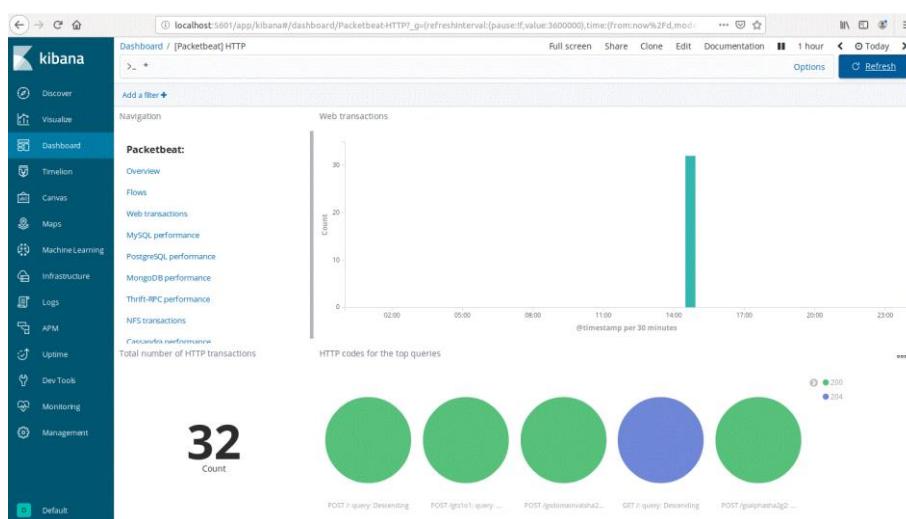


Figura 43: Packetbeat secuencia de ejecución

## 6.4.5 Auditbeat

Es básicamente una monitorización de seguridad, se encargará de recoger los eventos de sistema (Aplicación, seguridad, sistema) y enviarlos a Kibana.

Está basado en el servicio Auditd, que es un sistema de auditoría dentro de sistemas Linux, en el que podemos crear reglas personalmente, en estas reglas es donde indicamos que es lo que queremos monitorizar. Si vamos a nuestro archivo de configuración es donde podremos ver las diferentes reglas que hemos creado. Por defecto viene activado el módulo de integridad de archivo.

**Comprobamos que funciona.**

- 1) Editamos el Auditbeat.yml
- 2) Puesto que está activa la regla de monitorizar lo que se crean en los directorios más importantes entre ellos /bin/, creamos ahí un script inofensivo.
- 3) Kibana recogerá la información.

## 6.4.6 Libbeat

Por ultimo hacemos una pequeña mención a Libbeat que es la librería con la cual nos podemos configurar nosotros mismos las alertas, en nuestro caso esto no tendrá ningún sentido que las creáramos para este proyecto en particular, pero cabe destacar que todas las grandes compañías tienen su propia librería creada.

## 6.5 Logstash

### 6.5.1 Introducción

Hemos visto anteriormente como enviar eventos directamente de Beats a Elasticsearch, hablando en términos de Data Integration, en la ETL es como si hubiéramos tocado la **E** de Extracción y la **L** de Load y ahora pasáramos a ver la **T**.

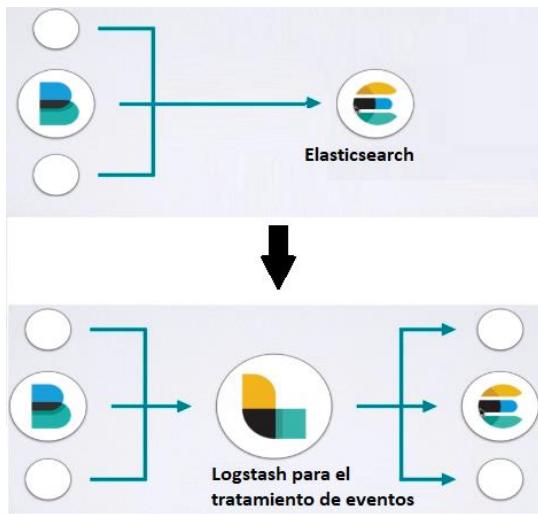


Figura 44: Tratamiento de eventos en Logstash

El input de Logstash va a venir en nuestro caso principalmente de Beat, pero este podría venir de un montón de servicios o bases de datos (Twitter,Github,Radis...).

**Grok:** Será una de las funciones principales que utilizaremos para especificar, que vamos a querer extraer de un evento, nos ayuda a identificar los diferentes campos, tiene funciones de etiquetado.

**GEOIP:** Un filtro interesante que nos proporciona Logstash es GEOIP que mediante geolocalización enriquece de información a la dirección Ip que llegan a nuestra infraestructura.

**Cifrado:** La comunicación es cifrada, tanto entrante en la parte de input como en la parte de output.

**Plugins:** Actualmente Logstash cuenta con unos 200 plugings tanto de entrada como de salida, pero es posible que queramos customizar un plugin que no existe, en este caso Logstash nos provee de toda la información necesaria para que nos creemos nuestro propio plugin para nuestro caso particular.

**Colas de almacenamiento:** Por último una propiedad interesante que nos ofrece Elastic es una cola de almacenamiento, esto es una especie de cache, es decir si nuestro servicio de output está caído por cualquier razón, se podrán guardar los datos de manera temporal en una cola de almacenamiento hasta que nuestro servicio vuelva a estar disponible. El tamaño de la cola está configurado por nosotros mismos, a más tamaño más tiempo podrá la cola aguantar recopilando datos mientras el servicio está caído.

## 6.5.2 Instalación y funciones

Una vez instalado Logstash en nuestro clúster podemos empezar a especificar la configuración, existen dos maneras de configurar las funciones, de forma manual y a través de archivo, para hacer una primera prueba vamos a hacer una primera ejecución de forma manual.

Le vamos a decir que todo lo que se encuentre en la línea de entrada, nos lo saque por la salida estándar. Simplemente lo que yo escribo por la entrada me lo saca por la salida.

```
test tf1
/usr/share/logstash/vendor/bundle/jruby/2.5.0/gems/awesome_print-1.7.0/lib/awesome_print/formatters/base_formatter.rb:31:
warning: constant ::Fixnum is deprecated
{
  "@version" => "1",
  "@timestamp" => 2019-09-13T08:48:02.006Z,
  "host" => "elastic04",
  "message" => "test tf1 "
}
```

Figura 45: Test y monitorización de Logstash

Movernos en el fichero de configuración va a ser mucho más fácil que crear nuestras reglas a mano por la pantalla.

Para crear nuestras reglas de monitorización nos meteremos en config. Hemos visto anteriormente que existen numerosos plugins de monitorización, en nuestro caso vamos a utilizar File. Ponemos nuestra primera regla utilizando el plugin – File. Escuchar los logs que escribamos dentro de un archivo.

La salida nos lo imprimirá por la salida estándar. Para probarlo, vamos a probar dicho archivo

```
{
  "@timestamp" => 2019-09-13T18:33:39.672Z,
  "size" => "3345k",
  "Data-Source" => "Salmon-230303",
  "Date_of_Ingest" => "20190913:20:24:32",
  "path" => "/home/elastic04/Documents/datos.json",
  "host" => "elastic04",
  "@version" => "1",
  "skript-Name" => "Scrapy"
}
```

Figura 46: Salida de Logstash

Esto es un log que monitoriza cuando han insertado nuestros scripts en la base de datos, tenemos la información específica que hemos recibido, la información está ya muy limpia pero por ejemplo en tamaño (size) queremos quitar la “k” para luego poder tratarlo como un número y así poder hacer la suma, media etc...

Para llevar esta tarea acabo necesitaremos el pluging de mutate, dentro de mutate necesitamos hacer uso de “gsub”, entonces adaptamos nuestro fichero con la siguiente configuración. Logstash nos imprime por pantalla las nuevas reglas.

```
[2019-09-14T10:50:51,051][INFO ][logstash.agent] Successfully started Logstash API endpoint {:port=>9600}
/usr/share/logstash/vendor/bundle/jruby/2.5.0/gems/awesome_print-1.7.0/lib/awesome_print/formatters/base_formatter.rb:31: warning: constant ::Fixnum is deprecated
{
    "path" => "/home/elastic04/Documents/datos.json",
    "Data-Source" => "Salmon-230304",
    "skript-Name" => "Scrapy_8389293824",
    "size" => "112",
    "@timestamp" => 2019-09-14T08:54:26.191Z,
    "host" => "elastic04",
    "Date_of_Ingest" => "20190913:20:24:32"
}
```

Figura 47: Primera transformación en Logstash

### Añadir campo

Según vaya evolucionando nuestra empresa la ingesta de base de datos no se basara solo en hacer web crowling de páginas webs sino, que por ejemplo queremos leer twitter, habrá volcados a nuestra base de datos, provenientes de otras fuentes etc...

En este caso nuestros datos han entrado a través de script de web-crawling y queremos marcarlos, para ello creamos una nueva regla que nos ayude en un futuro a poder identificar que volumen de datos han sido insertados mediante script.

Caso de uso: Imaginemos que en un futuro queremos, contar cuantos despliegues han entrado en nuestro sistema por medio de scripts y web-crawling y cuantos han sido por otros métodos.

**Problema:** Nuestro identificador único de script tiene un nombre muy largo y único, por lo que no se va a poder hacer una Query de count para obtener el número, lo que necesitaremos es añadir un nuevo campo que identifique si la entrada en el sistema ha venido a través de script o de otro método y lo indique en ese nuevo campo, así posteriormente podremos hacer unas consultas mas fáciles, esto se hace en logstast mediante.

```
root@elastic04:/etc/logstash/conf.d# echo '{ "Data-Source": "Salmon-230305", "skript-Name": "Scrapy_8389293824", "Date_of_Ingest": "20190913:20:24:32", "size": "112" }' >> "/home/elastic04/Documents/datos.json"
```



El campo Web-Crawler no existia antes, se ha creado

```
/usr/share/logstash/vendor/bundle/jruby/2.5.0/gems/awesome_print-1.7.0/lib/awesome_print/formatters/base_formatter.rb:31: warning: constant ::Fixnum is deprecated
{
    "Data-Source" => "Salmon-230305",
    "skript-Name" => "Scrapy_8389293824",
    "@timestamp" => 2019-09-14T09:23:57.675Z,
    "host" => "elastic04",
    "Ingest_Type" => "Web-Crawler",
    "size" => "112",
    "Date_of_Ingest" => "20190913:20:24:32",
    "path" => "/home/elastic04/Documents/datos.json"
}
```

Figura 48: Transformación en Logstash II

## Grok

La función Grok es una de las más importantes en Logstash es una sirve para descomponer en campos un log, hasta ahora habíamos insertado en modo JSON para hacer nuestras pruebas, sin embargo en un futuro es muy posible que los logs solo vengan como un mensaje de texto, Grok será capaz de descomponernos ese texto en diferentes campos.

## Geoip

Una de las funciones más importantes que tenemos es la de geolocalización por ip, en este caso.

```
[2019-09-14T15:26:16,990][INFO ][logstash.agent] Successfully started Logstash API endpoint {:port=>9600}
/usr/share/logstash/vendor/bundle/jruby/2.5.0/gems/awesome_print-1.7.0/lib/awesome_print/formatters/base_formatter.rb:31: warning: constant ::Fixnum is deprecated
{
  "src_geotip" => {
    "country_code3" => "CN",
    "ip" => "119.176.103.117",
    "location" => {
      "lon" => 116.9972,
      "lat" => 36.6683
    },
    "city_name" => "Jinan",
    "country_name" => "China",
    "longitude" => 116.9972,
    "continent_code" => "AS",
    "latitude" => 36.6683,
    "country_code2" => "CN"
  },
  "host" => "elasti04",
  "message" => " script->salmon-scrappy2233433 14092019 3442",
  "path" => "/home/elasti04/Documents/datos.json",
  "@version" => "1",
  "srcip" => "119.176.103.117",
  "@timestamp" => 2019-09-14T13:28:36.877Z
}
```

Figura 49: Activando la Geolocalización

## 6.5.3 Monitorizando nuestra infraestructura: Filebeat, Logstash, Elasticsearch.

Anteriormente en el primer apartado de Beats, habíamos visto como gracias a este servicio de ELK podíamos enviar información directamente a Elasticsearch, por otro lado en Logstash hemos visto cómo tratar logs, y hacer pequeñas transformaciones en ellos mediante diferentes reglas que escribimos en nuestro archivo de configuración, en este apartado vamos a empezar a monitorizar nuestra infraestructura con la combinación de ambos servicios, es decir por un lado vamos a activar Filebeat y le vamos a dar las instrucciones para que nos envíe esta información a Logstash, una vez allí escribiremos una serie de código para que podamos tratar ese Log y procesarlo más fácilmente en Elasticsearch como paso final será Kibana.

### Pasos a seguir:

- 1) Ir a filebeat.yml para modificar dicho archivo, indicando que nos envíe los logs a Logstash en vez de directamente a Elasticsearch.
- 2) Reiniciar el servicio de filebeat para que empiece a leer los logs con la nueva configuración.
- 3) Configurar el nuevo fichero en Logsh donde le vamos a indicar las nuevas reglas a tratar.  
Haremos uso de las siguientes funciones:

**Grok:** Para trocear el log e indicar cada campo.

**Geoip:** Para enriquecer nuestro log con información geolocalización

También deberemos crear un índice

```

input {
    #Indicamos que la entrada es el puerto 5044 que es donde envia Filebeat
    beats {
        port => 5044
    }
}
filter {
    grok {
        #hemos visto como grok, le podias configurar como iba a ser el mensaje , para que lo
        # troceara en diferentes campos, pero estavez "COMBINEDAPACHELOG" es un archivo
        # que ya viene con las instrucciones necesarias para hacer eso.
        match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
    date {
        match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
    geoip{
        source => "clientip"
    }
}
output {
    elasticsearch {
        # Indicamos que nos lo envie a elasticsearch
        hosts => ["localhost:9200"]
        # creamos un index propio para que no se confunda con los Filebeats
        index => "apache-%{+YYYY.MM.dd}"
    }
    stdout { codec => rubydebug }
}

```

Figura 50: Modificando el archivo Logstash.yml para hacer nuestra ETL

#### 4) Arrancar el servicio de Logstash

```
etc/logstash/conf.d/apache.conf --path.settings=/etc/logstash
```

#### 5) Generar logs en Apache con nuestro script generador de logs.

```
etc/logstash/conf.d/apache.conf --path.settings=/etc/logstash
```

#### 6) Generar los índices en Kibana para poder monitorizarlos

Comprobamos en nuestro servidor de Logstash que la información de localización esta llegando correctamente.

```

1,
  "ident" => "-",
  "source" => "/home/elasticsearch04/Downloads/Fake-Apache-Log-Generator/access_log_20190914-191601.log",
  "geoip" => {
    "latitude" => 37.5111,
    "country_code3" => "KR",
    "city_name" => "Seoul",
    "ip" => "121.166.45.42",
    "region_name" => "Seoul",
    "country_code2" => "KR",
    "longitude" => 126.9743,
    "region_code" => "11",
    "country_name" => "Republic of Korea",
    "location" => {
      "lon" => 126.9743,
      "lat" => 37.5111
    },
    "timezone" => "Asia/Seoul",
    "continent_code" => "AS"
  },
  "@timestamp" => 2019-09-14T17:27:43.000Z,
  "httpversion" => "1.0",
  "request" => "/apps/cart.jsp?appID=2929"

```

Figura 51: La localización está llegando correctamente

El siguiente paso es comprobar en Kibana que efectivamente ha llegado la información a Elasticsearch.

#### **6.5.4 Creado un mapa de coordenadas mediante la geolocalización: Mapping**

En el apartado anterior hemos activado la siguiente secuencia Filebeat -> Logstash -> Elasticsearch -> Kibana.

Ahora que ya tenemos unos datos enriquecidos con geolocalización lo que queremos es crear un mapa que lo podemos visualizar en un Dashboard. Este proceso se llevará a cabo mediante un “template”.

Pasos a seguir:

- 1) Nos vamos a Logstash y creamos una carpeta llamada “template”.
- 2) Dentro de la carpeta “template” creamos un archivo JSON con la configuración que necesitamos, en nuestro caso indicaremos los campos que vamos a querer que nos identifique, como la latitud, la longitud. Lo que nos interesa es sobretodo cambiar su data type para poder trabajar con ellos.

```

"properties": {
    "@timestamp": {
        "type": "date"
    },
    "bytes": {
        "type": "integer"
    },
    "geoip": {
        "properties": {
            "ip": {
                "type": "ip"
            },
            "location": {
                "type": "geo_point"
            },
            "latitude": {
                "type": "half_float"
            },
            "longitude": {
                "type": "half_float"
            }
        }
    }
}

```

Figura 52: Plantilla personalizada

### 3) Insertar nuestro template personalizado en los templates de elasticsearch.

```
curl -XPUT 'http://localhost:9200/_template/apache' -H 'Content-Type: application/json' -d@/etc/logstash/templates/apache_template.json
```

- 4) Comprobamos que nuestra plantilla existe realmente en elasticsearch. Hasta ahora habíamos utilizado siempre el comando GET \_cat/Indices/\* para obtener los diferentes índices que teníamos, ahora lo que necesitamos es ver los esquemas, y comprobar si realmente nuestro esquema se ha insertado con éxito.

```

+ GET _template/apache
  32   },
  33     "match_mapping_type" : "string"
  34   },
  35   {
  36     "strings_as_keyword" : {
  37       "mapping" : {
  38         "ignore_above" : 1024,
  39         "type" : "keyword"
  40       },
  41       "match_mapping_type" : "string"
  42     }
  43   },
  44   ],
  45   "properties" : {
  46     "@timestamp" : {
  47       "type" : "date"
  48     },
  49     "geoip" : {
  50       "properties" : {
  51         "ip" : {
  52           "type" : "ip"
  53         },
  54         "latitude" : {
  55           "type" : "half_float"
  56         },
  57         "location" : {
  58           "type" : "geo_point"
  59         },
  60         "longitude" : {
  61           "type" : "half_float"
  62         }
  63       },
  64       "bytes" : {
  65         "type" : "integer"
  66       }
  67     }
  68   }
}

```

Figura 53: Plantilla de Apache

Todo evento que indexemos a partir de ahora en apache\* lo que va a hacer es matchear nuestro esquema que hemos predefinido y los datos van a tener el tipo que hemos especificado.

### 5) Borrar nuestro antiguo índice tanto de Kibana como de Elasticsearch.

Un problema que nos encontramos es que, si anteriormente habíamos indexado una serie de campos con una data tipo, y nos lo había reconocido como fue en nuestro caso como tipo string, no podemos cambiar el tipo sobre la marcha. En nuestro caso la única solución que nos encontramos es tener que borrar el índice apache que teníamos anteriormente.

Finalmente podemos obtener la geolocalización de nuestras entradas de Ip

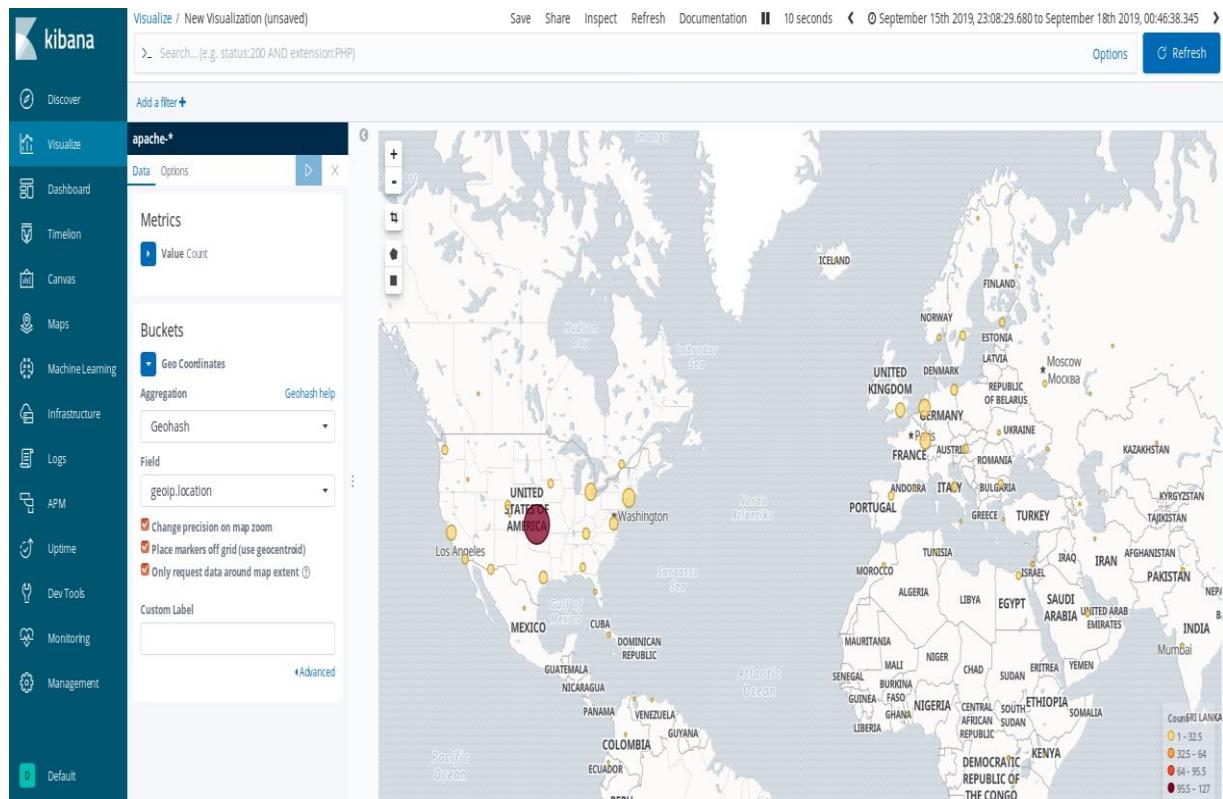


Figura 54:Logs de entrada de apache, dibujados en un mapa

# Capítulo 7. Visualización de datos, Kibana

## 7.1 Introducción

Kibana es el elemento en la pirámide de ELK, hemos visto Elasticsearch y Logstash en profundidad y aunque ya hemos visto Kibana en este apartado ahondamos aún más en esta herramienta tan potente que nos proporciona Elasticsearch.

Kibana actúa parecido a como lo hacen otras herramientas de visualización como son Tableau, QlikSense etc.. con la diferencia de que esta herramienta tiene mucho potencial en el procesado de Logs.

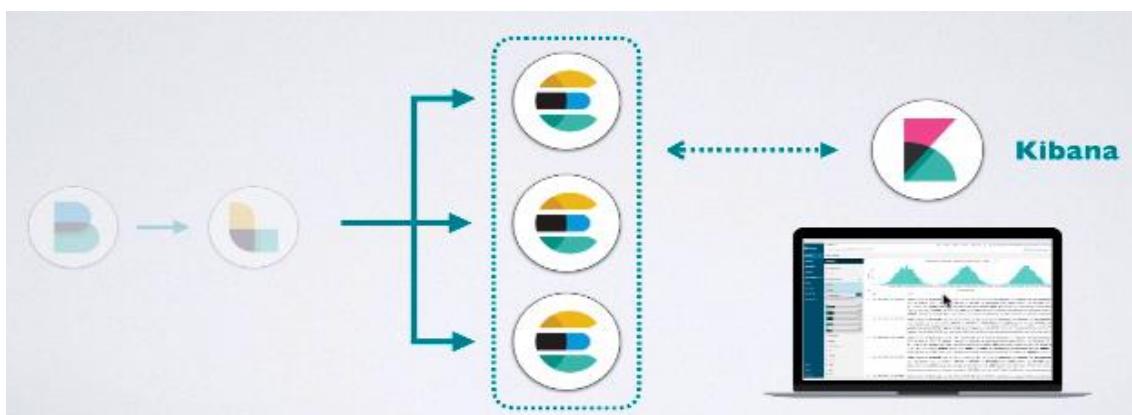


Figura 55: ELK-Kibana

## 7.2 Discover

Es una forma de ver cada uno de los eventos relacionados, se le puede aplicar consultas y filtros. Es el más utilizado, se le debe indicar que índice quiere visualizar, y se puede interactuar con los intervalos de tiempo para indicar cada cuanto queremos que nos muestre la información. Se puede ajustar cada cuanto tiempo deseamos hacer el refrescado de datos, se puede interactuar con la pantalla y mirar un cierto periodo de tiempo específico.

## Arquitectura Big Data: Pipeline y Monitorización

Enrique Benito Casado

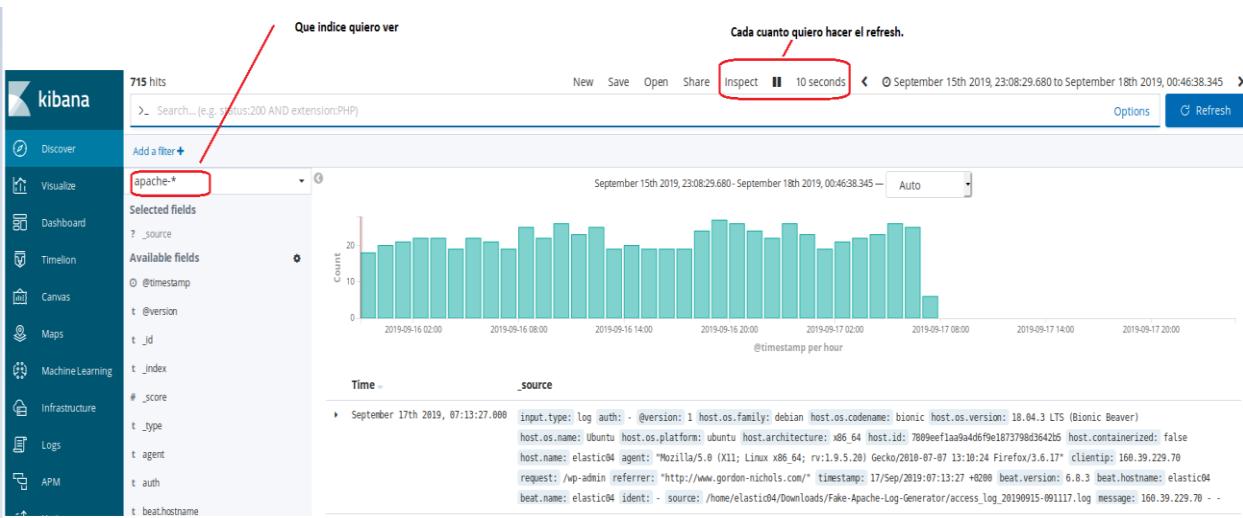


Figura 56: Visualizando nuestros logs en discover I

Los campos se desplegaran pinchando en cada uno de los eventos.

Discover provee arriba del todo un buscador donde podremos realizar consultas, sobre nuestro indice, por ejemplo arriba tenemos el campo de geolocalizacion geoip.city\_name:Beijing, nos devolveria todos los datos que matchearan con nuesra query dada. Ademas nos permite incluir cierta logica, es decir podriamos meter condiciones como AND.

Ejemplo: response:200 AND geoip.city\_name:Beijing

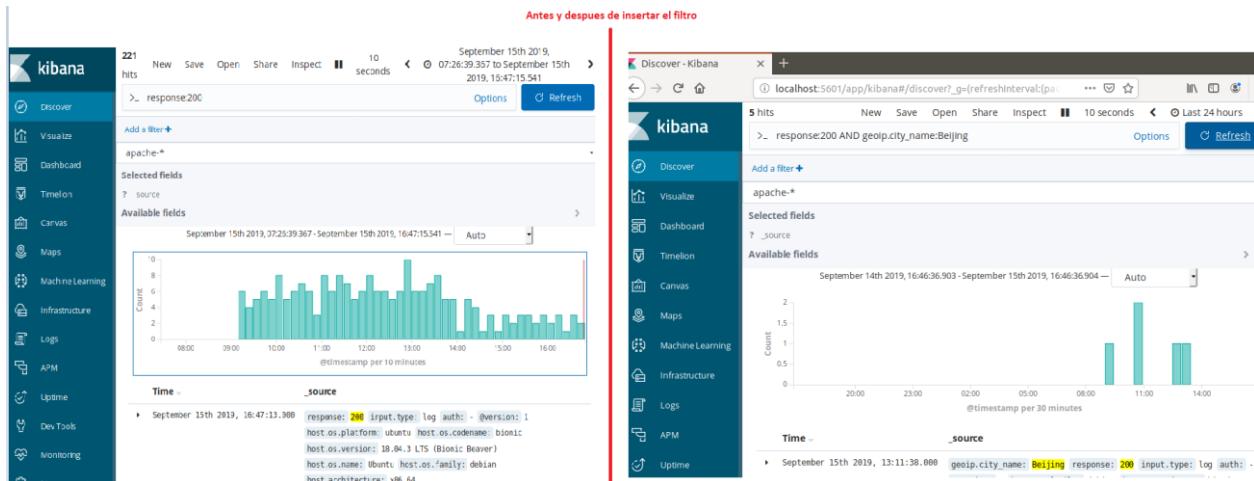


Figura 57: Añadiendo un filtro a nuestra visualización

## 7.3 Visualizaciones y Dashboards

Al final es lo que vamos a buscar, mostrar nuestra información de la manera más óptima posible. Es uno de los módulos más importantes de ELK.

Existen numerosos tipos de Dashboards, y podríamos dedicarle un capítulo entero, pero vamos a centrarnos en los más básicos. Un dashboard se compone de diferentes tipos de visualizaciones por lo que en este apartado lo que hacemos es crear diferentes tipos de visualizaciones y unirlas de manera que en un simple vistazo podamos verlas todas.

Primera visualización:

Histograma de eventos: Donde la agregación viene por medio de Date Histogram, además queremos cortar nuestras barras por el campo “response” para ver así qué tipo de respuesta se ha ido dando. Una vez hayamos realizado esta visualización debemos guardarla, para utilizarla en un futuro.

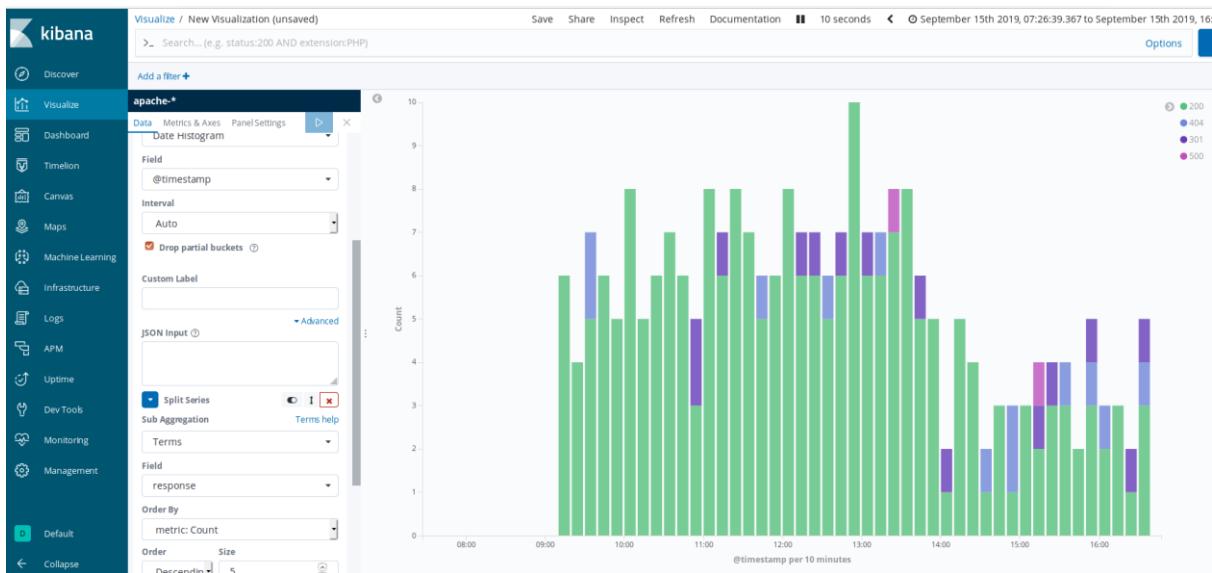


Figura 58:Barchar cortado por response

### Métrica

Lo siguiente que vamos a elegir va a ser una métrica, en concreto en nuestro caso queremos ver el número de IP únicas que nos llegan a nuestro servidor. Una vez tenemos la métrica la guardamos igual.

La métrica se constituye de un valor numérico que podremos insertar en algún lugar de nuestro Dashboard.

### Visualización Donuts

Crearemos una visualización en forma de Donuts para visualizar en conjunto las respuestas que estamos obteniendo, de esta manera tenemos una visualización más clara del conjunto general de respuestas que estamos obteniendo.

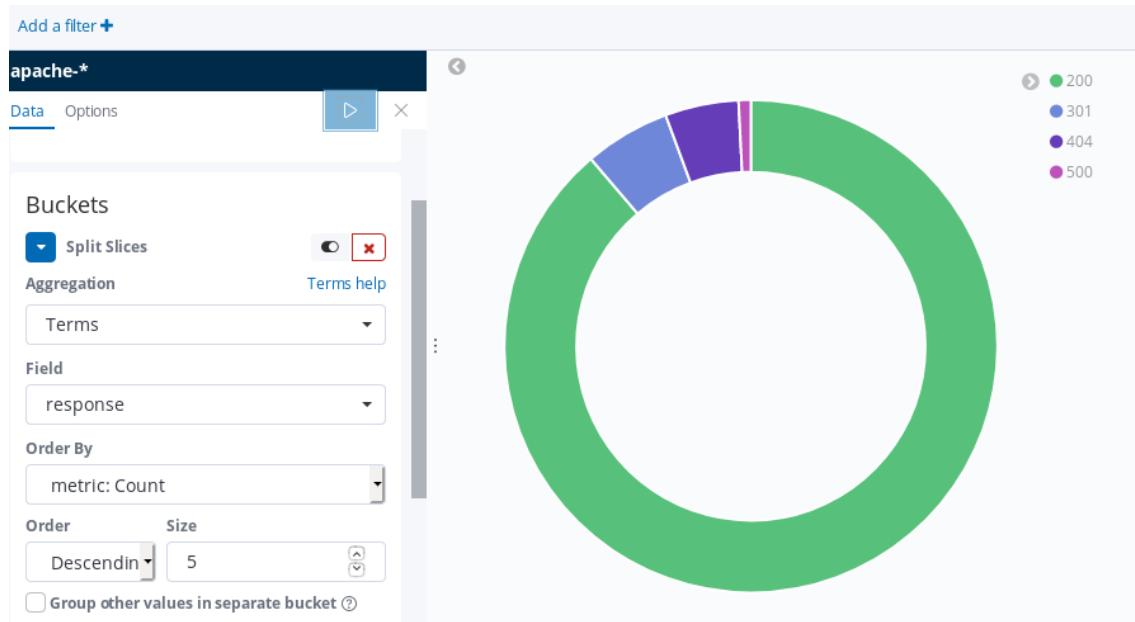


Figura 59: Visualización Donuts

### Tabla

Creamos una visualización en tabla donde mostramos un ranking una tabla con la información de las Ips de los países que nos han visitado.

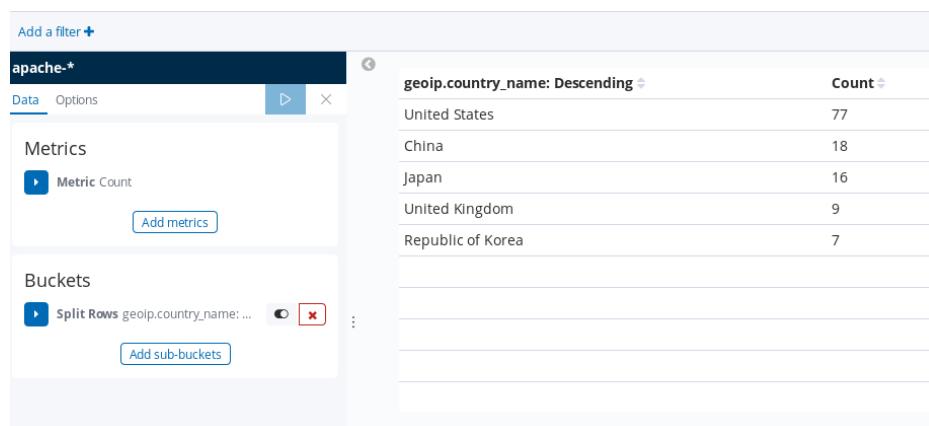


Figura 60: Visualización tabla

## Montando nuestro Dashboard

Una vez tenemos los gráficos que queremos visualizar, es el momento de construir nuestro Dashboard.

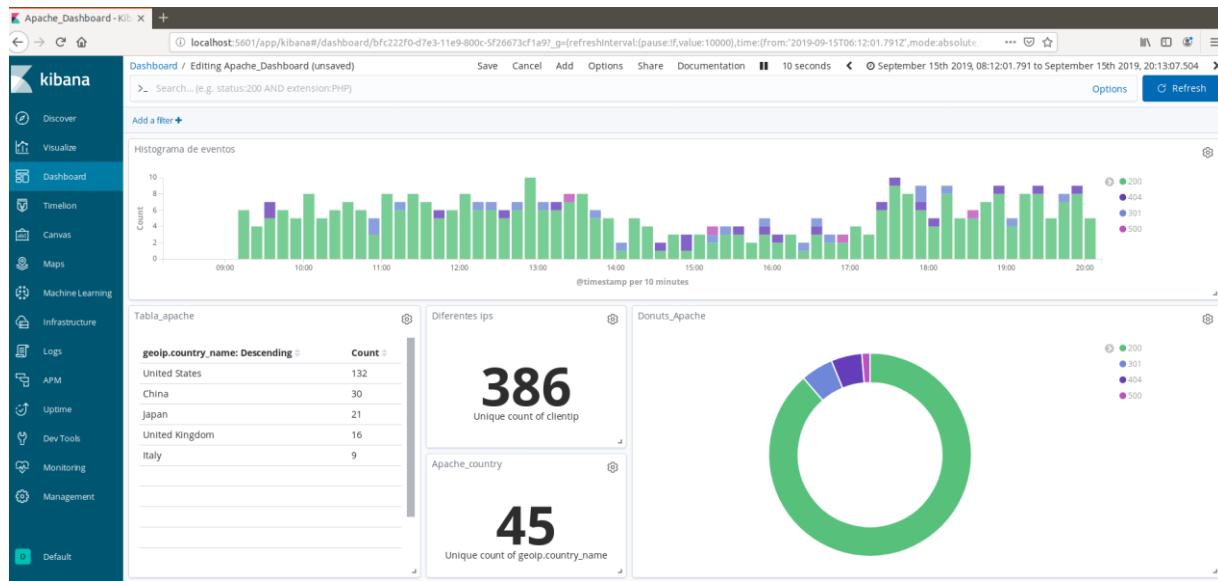


Figura 61: Uniendo Todo, dashboard final

## 7.4 APM

Es uno de los últimos módulos en integrarse en ELK y nos monitoriza el rendimiento de aplicaciones. En nuestra infraestructura Big Data, nuestro equipo de desarrollo que se encarga de aplicar ML (NLP) a los documentos de texto, puede instalarse este módulo para monitorizar el rendimiento de sus programas, es decir es una monitorización interna del código que estamos creando. En este caso solo lo mencionamos y no trabajamos con ello, sino que lo dejamos para líneas futuras.

# Capítulo 8. Machine Learning - X-PACK

Este módulo es en Elastic un módulo de pago, por lo que en este TFM solo se va a poder poner información Teórica.

## 8.1 Machine Learning sobre los logs del servidor

Analiza un comportamiento y aprende de el, todo lo que se salga de este comportamiento se trataría como una anomalía.

**Machine Learning aplicado al servidor.**

Pasos a seguir.

- 1- Como no disponemos de millones de logs de servidor, los importamos, elastic nos proporciona ya unos datos para hacer las pruebas.

Código:

```
wgethttps://download.elasticsearch.org/demos/machine_learning/gettingstarted/servermetrics.tar.gz
```

- 2- Creamos el índice Servidor-metrics mediante una plantilla personalizada, esto es parecido a Create table de SQL donde se dice que campos tendrá la tabla, en este caso se indica la estructura del índice.

Código: curl -X PUT "localhost:9200/servermetrics" -H 'Content-Type: application/json'  
-d' → aquí metemos el schema.

- 3- Lo siguiente que nos queda es lanzar el script de subida de datos que nos proporciona Elasticsearch

```
root@elastic04:/home/elastic04/Downloads/server_metrics/files# sh upload_server_metrics.sh

== Script for creating index and uploading data ==

== Deleting old index ==

{"acknowledged":true}
== Creating Index - server-metrics ==

{"acknowledged":true,"shards_acknowledged":true,"index":"server-metrics"}
== Bulk uploading data to index...

Server-metrics_1 uploaded
Server-metrics_2 uploaded
Server-metrics_3 uploaded
Server-metrics_4 uploaded
```

Figura 62: Subiendo a elasticsearch nuestra Data

Los datos que hemos subido son peticiones a un servidor, contamos con millones de archivos, huelga decir que un humano sería incapaz de sacar patrones o outliers, es por eso que queremos ver si mediante Machine Learning se puede identificar.

Al creares el job empieza automáticamente a analizar los datos, y vemos que nos devuelve el siguiente resultado, indicando que ha detectado outliers.

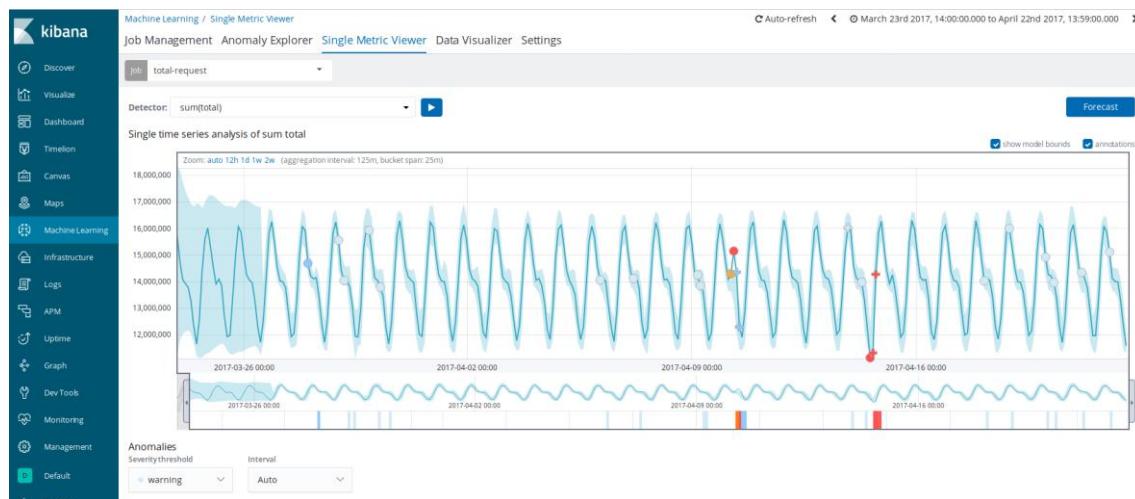


Figura 63: Aplicando ML a nuestros logs

Haciendo zoom en la zona señalada, vemos más de cerca la anomalía, mediante ML había identificado un patrón de comportamiento que no se ha dado.

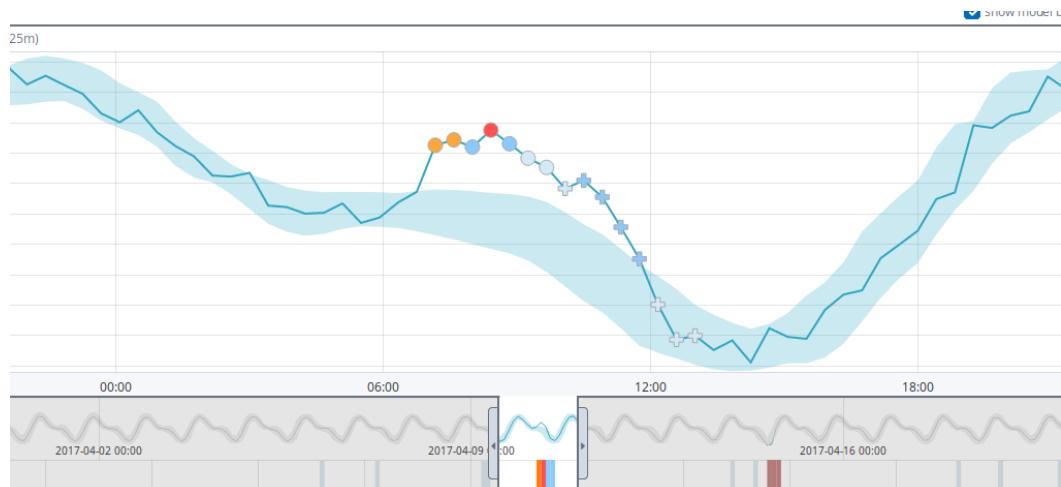


Figura 64: Observando de cerca las anomalías

## 8.2 ML aplicado a actividades de usuario.

Vamos a aplicar ML a las actividades que los usuarios están teniendo en nuestra infraestructura. En este caso vamos a crear un job avanzado que nos proporcionara la posibilidad de añadir un detector.

Pasos a seguir:

- 1- Volvemos a descargarnos datos que nos ofrece elastic.

Codigo:

```
https://download.elasticsearch.org/demos/machine_learning/gettingstarted/user-activity.json
```

wget

- 2- Indexamos en elasticsearch.

Codigo: `curl -s -X POST -H "Content-Type: application/json" localhost:9200/user-activity/_bulk --data-binary "@user-activity.json"`

Los datos representan la cantidad de Bytes que los usuarios están enviando, por tanto en este caso el detector que vamos a crear le diremos que actúe sobre la media de los bytes que los usuarios envían a nuestro servidor y vamos a buscar alguna anomalía, el cual sigue un comportamiento habitual, de repente tenga un comportamiento inusual.

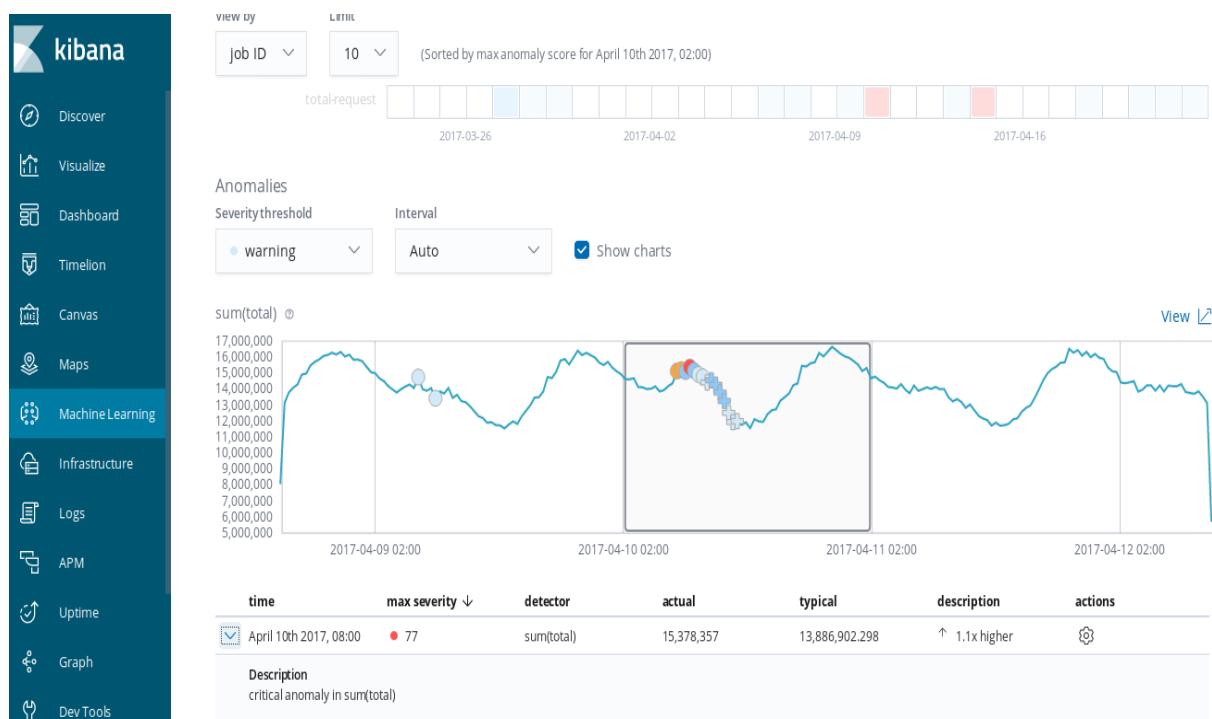


Figura 65: Aplicando ML a métricas de usuarios

# Capítulo 9. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

## Infraestructura

**Migración a la nube:** Nuestro proyecto ha sido montado mediante un ordenador y la ayuda de diferentes máquinas virtuales, se ha pretendido hacer una simulación de una infraestructura Big Data On-Premise, el siguiente paso lógico es migrar toda nuestra infraestructura de On-Premise a la nube, virtualizando nuestros servicios mediante contenedores Docker y orquestándolos mediante Kubernetes.

**Jobs-Scheluder/Orquestador:** Sin duda es un punto muy importante que le falta a este TFM la falta de un orquestador general o un software que programe que Jobs van a ser lanzados, y en qué orden. Existen diferentes programas de orquestación como Uc4, Autosys o incluso Apache Airflow que es opensource, por temas de tiempo no ha sido posible contar con estas herramientas.

**Data Streaming:** Sin duda, un tema muy interesante donde se puede profundizar mucho, especialmente interesante aplicado a plataformas como twitter. Apache-Stream o Apache Flink son dos de las herramientas más interesantes en este campo.

## Data Science

**NLP:** Análisis de sentimiento sobre el texto. Para poder comparar predicciones, producto y bolsa.

**Evitar fraude:** Los autores “actualizan” sus textos, esto puede provocar que un autor que había hablado negativamente sobre un producto cambie su texto en función de cómo evoluciona el mercado.

## Capítulo 10. Referencias

### Herramientas y programas utilizados

Scrapy: <https://scrapy.org/>

MongoDB: <https://www.mongodb.com/download-center/community>

Pymongo: <https://api.mongodb.com/python/current/>

Apache Kafka: <https://kafka.apache.org/downloads>

Elasticsearch: <https://www.elastic.co/de/downloads/elasticsearch>

Logstash: <https://www.elastic.co/de/downloads/logstash>

Beats: <https://www.elastic.co/de/products/beats>

Filebeat: <https://www.elastic.co/guide/en/beats/filebeat/master/filebeat-getting-started.html>

Auditbeat: <https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-installation.html>

Metricbeat: <https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-installation.html>

Kibana: <https://www.elastic.co/de/downloads/kibana>

Virtualbox: <https://www.virtualbox.org/wiki/Downloads>

Ubuntu: <https://ubuntu.com/download/desktop>

Fake-Apache-Log-Generator: <https://github.com/kiritbasu/Fake-Apache-Log-Generator>

## BIBLIOGRAFÍA

*Todd Palino, Gwen Shapira, Neha Narkhede: “Kafka: The definitive Guide”.*

*Ryan Michael : “Web Scraping with Python: Collecting More Data from the Modern Web”.*

*Radu Gheorghe, Matthew Lee Hinman, and Roy Russo: “Elasticsearch in Action”.*

**Eric Redmon y Jim Wilson: “Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement”.**

*Martin Fowler: “NoSQL Distilled”.*

Documentacion de kafka: <https://www.confluent.io/>

Documentacion de Elasticsearch: <https://www.elastic.co/guide/index.html>

MongoDB vs Elasticsearch: <https://medium.com/@ranjeetvimal/elasticsearch-vs-mongodb-631f410cd317>