ElasticSearch.

Contenido

[Recursos útiles 1](#_Toc203945961)

[Objetivo de la práctica 2](#_Toc203945962)

[Instalación MongoDB y herramientas asociadas 2](#_Toc203945963)

[Instalación Elasticsearch como Docker 2](#_Toc203945964)

[Clientes API REST 3](#_Toc203945965)

[Práctica con el Quijote 3](#_Toc203945966)

[Carga del documento 3](#_Toc203945967)

[Búsquedas 4](#_Toc203945968)

[Tabla de consultas básicas en Query DSL 5](#_Toc203945969)

[Uso desde Python 6](#_Toc203945970)

[Script Python. 6](#_Toc203945971)

[Exploración y consultas desde Kibana 6](#_Toc203945972)

[Visualización de datos con Discover 7](#_Toc203945973)

[Consola de consultas DSL con Dev Tools 7](#_Toc203945974)

[Visualización rápida con Visualize Library 8](#_Toc203945975)

[Entregables de la práctica 9](#_Toc203945976)

# Recursos útiles

Recuerde el directorio de recursos compartidos disponible en:

<https://nascorformacion0-my.sharepoint.com/:f:/g/personal/juan_pinuela_docente_nascorformacion_com/EtGSBITnqFpJp6-XpG6REOQBFycP_G9zgcB89jxPymjV2Q?e=IBIEER>

Para esta práctica disponemos de una carpeta con nombre “**elk”** donde podemos encontrar todos los ficheros utilizados y creados en este guion.

Curso breve:

<https://www.youtube.com/playlist?list=PLlNuLwK_vaGND4FMd7E55kM8JG7JpCgA0>

# Objetivo de la práctica

Esta práctica tiene como objetivo introducir al alumno en el ecosistema **Elastic Stack (ELK)**, focalizando el trabajo en **ElasticSearch** como motor de búsqueda y análisis. A través de ejemplos prácticos, se pretende:

* Comprender el funcionamiento básico de ElasticSearch y su API REST.
* Aprender a cargar y consultar datos mediante índices.
* Utilizar herramientas como Postman (o Insomnia) para interactuar con ElasticSearch.
* Aplicar consultas en lenguaje Query DSL.
* Integrar ElasticSearch en un flujo de trabajo Python como parte de un sistema de información.

# Instalación MongoDB y herramientas asociadas

## Instalación Elasticsearch como Docker

Creamos la carpeta “C:\Docker\elk” donde copiamos el siguiente docker-compose.yml, que podemos lanzar sin más que ejecutando desde la línea de comandos:

>> docker-compose up -d

version: '3.8'

services:

elasticsearch:

image: docker.elastic.co/elasticsearch/elasticsearch:8.11.1

container\_name: elasticsearch

environment:

- discovery.type=single-node

- xpack.security.enabled=false

ports:

- 9200:9200

- 9300:9300

kibana:

image: docker.elastic.co/kibana/kibana:8.11.1

container\_name: kibana

ports:

- 5601:5601

environment:

- ELASTICSEARCH\_HOSTS=http://elasticsearch:9200

## Clientes API REST

Existen muchas herramientas para interactuar con ElasticSearch de entra las cuales nosotros usaremos Postman pues ya la tenemos instaladas de prácticas anteriores. En el vídeo de referencia se usa Insomina y en la siguiente gráfica se muestra nada menos que otras 14 alternativas cada una con sus características únicas pero no significativas para la funcionalidad básica de interactuar con un API REST.



# Práctica con el Quijote

## Carga del documento

Disponemos en los recursos de la unidad del fichero “**bulk\_quijote.json**”. Si analizamos su estructura, se trata de un archivo de inserción masiva JSON (bulk API) diseñado para:

* Crear documentos en el índice donquijote.
* Estructurar el texto del Quijote como una colección de líneas indexadas individualmente.

Podemos cargar el fichero con el API endpoint \_bulk, como se muestra en la siguiente imagen:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez cargado podemos consultar rápidamente cualquier línea o fragmento de texto permitiéndonos mostrar las capacidades de ElasticSearch para realizar búsquedas por texto, palabras, capítulo, etc.

## Búsquedas

Una vez cargado el documento para hacer búsquedas utilizaremos el lenguaje Query DSL cuya referencia podemos consultar en:

<https://www.elastic.co/docs/explore-analyze/query-filter/languages/querydsl>

Query DSL es un lenguaje de consulta de estilo JSON con todas las funciones que permite búsquedas, filtrado y agregaciones complejas. Es el lenguaje de consulta original y más potente para Elasticsearch en la actualidad.

Con el índice del Quijote, se propone construir queries para responder a las siguientes preguntas:

1. **¿En cuántas líneas aparece el nombre “Quijote”?**

Podemos hacerlo directamente con un GET al endpoint “\_search”

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

O con un JSON de búsqueda a ese mismo *endpoint*. como mostramos en la imagen siguiente, donde hemos usado el JSON siguiente:

{"query": { "match": {"text": "quijote"}}, "size": 0}

Texto

El contenido generado por IA puede ser incorrecto.

Al hacer “size”de valor 0, lo que estamos indicando es que **no queremos** que ElasticSearch devuelva documentos en los resultados. La consulta solo devolverá los metadatos, como el número total de coincidencias (hits.total).

1. **¿En cuántas Sancho? ¿Y juntos?**
2. **¿Cuántas líneas están vacías?**
3. **¿En algún capítulo no se habla del Quijote?**
4. **¿Cuántas líneas hay de menos de 10 palabras (pero al menos 1) en los capítulos 200, 201 y 202?**

## Tabla de consultas básicas en Query DSL

|  |  |  |
| --- | --- | --- |
| Tipo de consulta | Descripción | Ejemplo básico JSON |
| match | Busca términos analizados en un campo. | "match": { "text": "quijote" } |
| match\_phrase | Busca frases exactas (orden y palabras exactas). | "match\_phrase": { "text": "don quijote" } |
| term | Coincidencia exacta sin analizar el término. | "term": { "chapter": 1 } |
| range | Búsquedas por rangos (números, fechas, etc.). | "range": { "words": { "gte": 10, "lte": 20 } } |
| bool | Combinación lógica de varias consultas. | {"bool": { "must": [...], "should": [...], "must\_not": [...] }} |
| exists | Busca documentos que tengan un campo. | "exists": { "field": "text" } |
| prefix | Busca valores que empiezan por un prefijo. | "prefix": { "text": "quij" } |
| wildcard | Busca con comodines (\* y ?). | "wildcard": { "text": "quijot\*" } |
| multi\_match | Busca el término en varios campos a la vez. | "multi\_match": { "query": "quijote", "fields": ["text", "title"] } |

# Uso desde Python

Siguiente con nuestro flujo de trabajo ya iniciado con MongoDB y Neo4j de un pequeño sistema de información de productos y clientes vamos ahora a insertar los productos definidos e insertados en MongoDB como documentos de un índice de ElasticSearch desde Pyhton.

## Script Python.

Se dispone de buscar\_productos.py para realizar las siguientes acciones:

* Crea un índice llamado productos.
* Inserta los documentos desde productos.json usando el script buscar\_productos.py.
* Buscar productos por descripción con match.

Se pide al alumno comparar esta búsquedas entre el uso de Postman y el de Python para poder luego **realizar al menos un ejemplo de estas otras búsquedas**:

1. **Filtrar por categoria**
2. **Buscar productos con precio en un rango**
3. **Agregaciones: total por categoría, promedio de precios, etc.**

# Exploración y consultas desde Kibana

Además del trabajo con ElasticSearch mediante clientes API REST (Postman o Insomnia) y scripts Python, en esta práctica incorporamos el uso de **Kibana**, interfaz gráfica oficial del ecosistema Elastic Stack, desplegada automáticamente con nuestro Docker Compose. Una vez desplegado, podemos acceder a Kibana en:

<http://localhost:5601/>

## Visualización de datos con Discover

Desde el menú lateral, entra en Discover y selecciona el índice asociado al "Quijote". Explora los documentos, observa los distintos campos “text, words, chapter, etc” y practica aplicando filtro desde la interfaz gráfica.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Se pide al estudiante realizar una captura que evidencie como se pueden

1. **Localizar todas las líneas que contienen la palabra "Quijote" directamente desde la interfaz Discover.**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

## Consola de consultas DSL con Dev Tools

Con Dev Tools tenemos otro entorno similar a Postman para todo tipo de interacciones con el API de ElasticSearch. Por ejemplo podemos ver que índices tengo en un momento dado:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Como ya hemos practicado con Postman suficientes consolas por lo que se sólo se pide:

1. **Ejecutar la siguiente DSL desde Dev Tools y visualizar el resultado:**

GET donquijote/\_search

{

"query": {

"match": {

"text": "Quijote"

}

..

## Visualización rápida con Visualize Library

Aquí mostramos como hemos creado una visualización para mostrar la contribución relativa del precio de cada producto.

Gráfico, Gráfico circular

El contenido generado por IA puede ser incorrecto.

O las veces que aparece la palara Quijote

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

1. **Desde Visualize Library, crea una visualización básica para visualizar cómo se distribuyen las líneas del texto a lo largo de los capítulos.**

* Tipo: Gráfico de barras.
* Datos: Número de líneas (count) agrupadas por chapter.
* Guarda el gráfico como “Líneas por capítulo”.

# Entregables de la práctica

El alumno deberá entregar las siguientes evidencias al finalizar la práctica:

* **Capturas de pantalla** que evidencien:
  + Despliegue de ElasticSearch y Kibana (docker-compose).
  + Uso de Postman para ejecutar peticiones \_bulk y consultas.
  + Resultados de consultas (incluidas búsquedas por términos, líneas vacías, etc.) y uso de Kibana
* **Script Python**:
  + Archivo buscar\_productos.py con el código utilizado para:
    - Creación del índice productos.
    - Inserción desde productos.json.
    - Ejecución de búsquedas por descripción, categoría, rango de precios y agregaciones.
  + Capturas que evidencien la ejecución del script y resultados obtenidos.