

# Repositorios de Información

Una introducción “gentil” a  
ElasticSearch

# Aviso para navegantes

- Elasticsearch (ES) permite la creación de sistemas de recuperación de información escalables y distribuidos.
- Típicamente ES se monta en un clúster con múltiples nodos, cada nodo tendrá múltiples *shards* (fragmentos) de un índice, *shards* que, además, pueden tener múltiples réplicas.
- No vamos a ver esas características de ES (aunque podéis explorarlas opcionalmente en la práctica)
- Vamos a usar ES como una herramienta conveniente para entender algunas de las cuestiones más básicas en recuperación de información: indexado, consultas y relevancia.

# Lo primero es lo primero

- Vete a la página <https://www.elastic.co/products>
- Asegúrate de que estás en la “GA Release”
- Descarga el ZIP en tu directorio de usuario
- Descomprime el archivo ZIP
- Deberías tener la siguiente estructura de archivos y directorios:
  - `./bin`
  - `./config`
  - `./lib`
  - `./modules`
  - `./plugins`
  - `LICENSE.txt`
  - `NOTICE.txt`
  - `README.textile`
- Abre una ventana de comandos y ejecuta `elasticsearch.bat`
- En tu casa es mejor que lo instales como un servicio en Windows o, en Linux, que lo instales como un paquete más.
- Si en un navegador abres la URL <http://127.0.0.1:9200/> debería salir algo así...

# Lo primero es lo primero

Salida típica de <http://127.0.0.1:9200/>

```
{
  "name" : "FWgeCaC",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "hc9UIjb9RC6hU6k5TTlBIQ",
  "version" : {
    "number" : "5.6.1",
    "build_hash" : "667b497",
    "build_date" : "2017-09-14T19:22:05.189Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

# Comandos de Elasticsearch

- Los comandos de ES toman parámetros en formato JSON y se invocan mediante HTTP (**GET**, **POST**, **PUT**, **DELETE**).
- Ventaja:
  - Puede “atacarse” ES desde cualquier lenguaje de programación que permita consumo de servicios web REST.
- Inconvenientes:
  - Utilizar ES a “bajo nivel” (sin interfaz o lenguaje de programación) es tedioso e impráctico.
  - Cualquiera con acceso a la URL del clúster ES puede ejecutar comandos (p.ej. Borrar un índice completo).
    - Existen plugins de autenticación y seguridad para ES

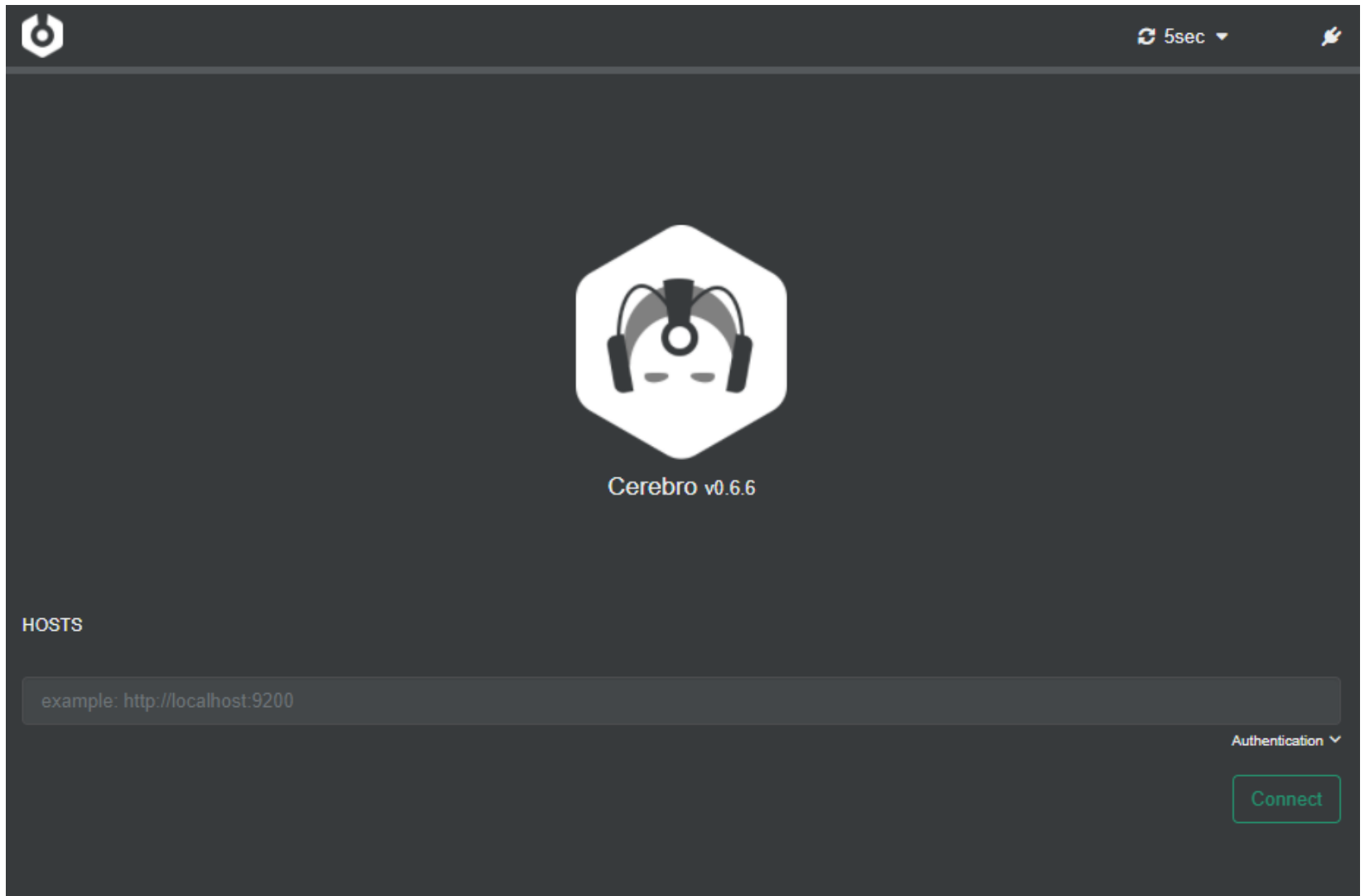
# Instalando un front-end para administrar Elasticsearch

- Antes de ES 5 existían (siguen existiendo) varios plugins como Head, Kopf o HQ.
- Desde la versión 5 dichos plugins ya no son servidos por Elasticsearch sino que deben servirse con otro servidor web.
- En lugar de usar uno de esos plugins y “cacharrear” para hacerlo funcionar vamos a instalar una herramienta nueva: cerebro

# Instalando cerebro

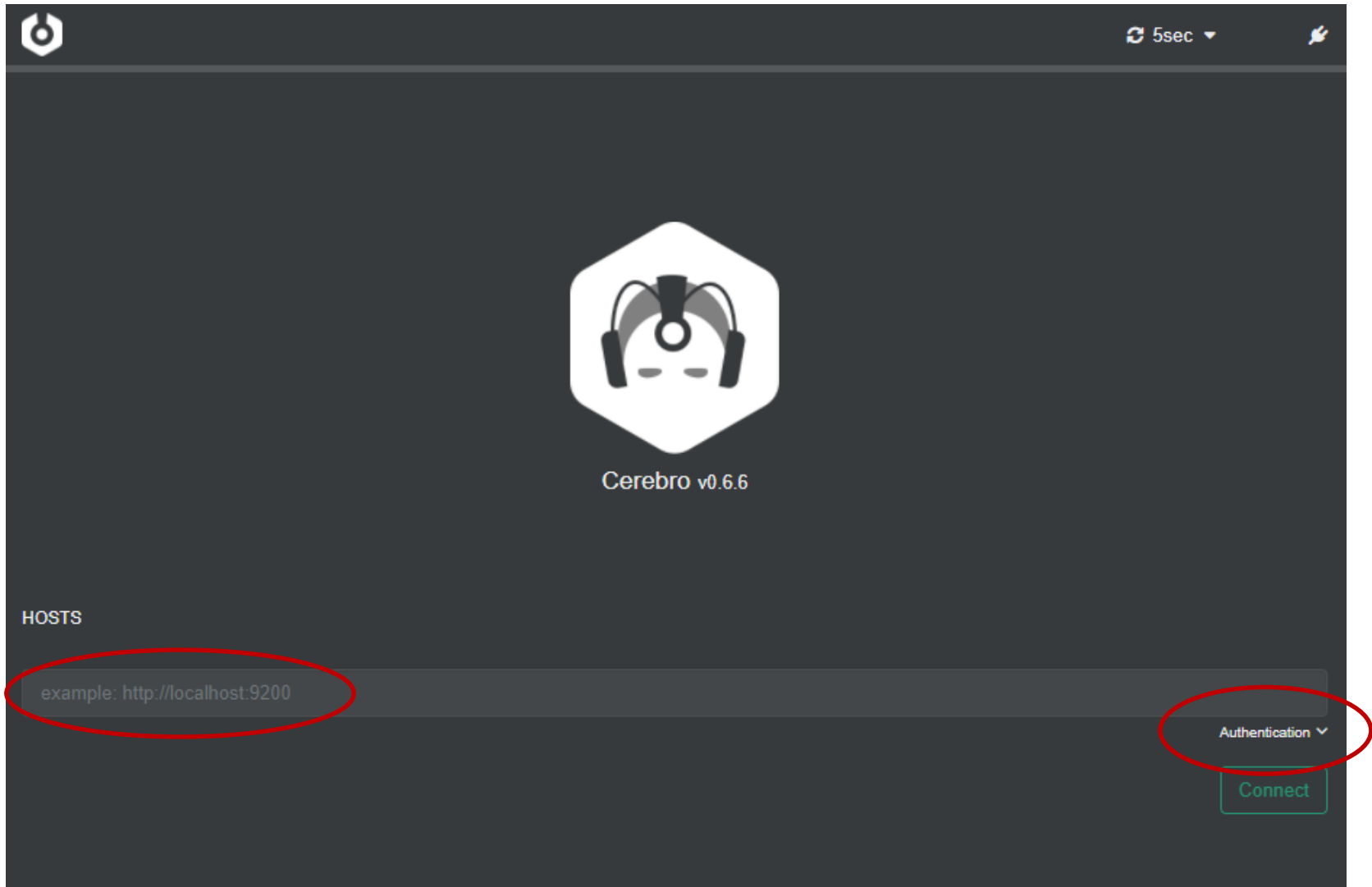
- Vete a la página web <https://github.com/lmenezes/cerebro/releases>
- Descarga en tu directorio de usuario la última versión de cerebro en formato ZIP (no necesitas el código fuente)
- Descomprime el archivo. Deberías tener la siguiente estructura de archivos y directorios:
  - `./bin`
  - `./conf`
  - `./lib`
  - `README.md`
- Abre una ventana de comandos y ejecuta `./bin/cerebro`
- Vete a la URL <http://127.0.0.1:9000>
- Debería aparecer una página tal que así...

# Pantalla de inicio de cerebro

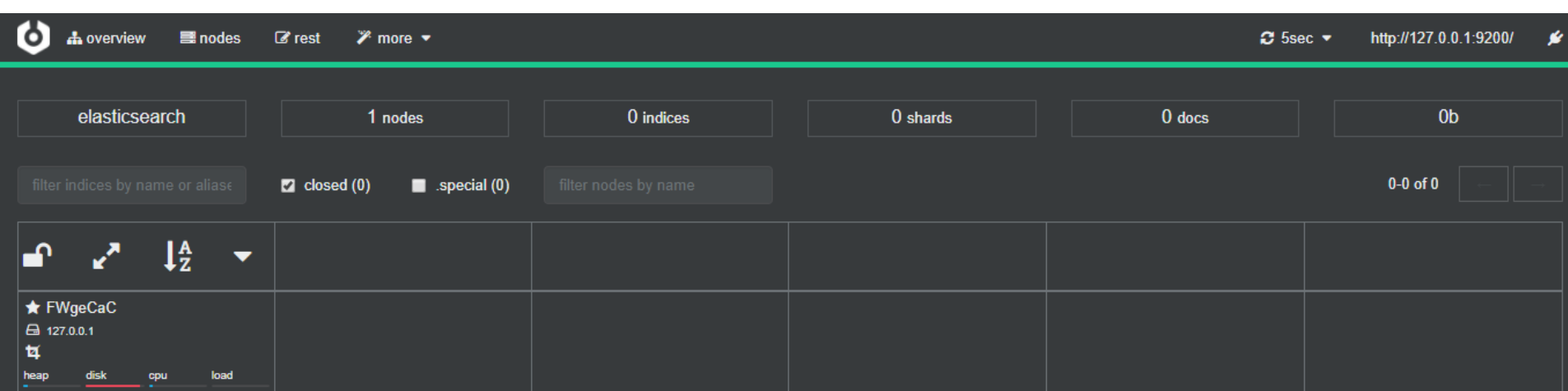




# Pantalla de inicio de cerebro



# Pantalla principal de cerebro



# Hora de crear un índice

- El índice de ES es lo que en teoría habíamos llamado **“fichero invertido”**
- Para construirlo necesitamos, obviamente, una **colección de documentos**.
- ES puede indexar cualquier tipo de documentos pero siempre hay que transformarlos en JSON
  - Siempre puede ser un documento JSON con un único campo cuyo contenido sea el texto del documento.
- Por simplicidad vamos a descargar una colección que ya consiste en documentos JSON.

# La colección de documentos

- Vete a la URL <http://goo.gl/aAtVmm>
- Descarga y desomprime el archivo  
**2008-Feb-02-04.json.gz**  
(serán 221 MB y 700.831 tuits)
- El archivo contiene **todos** los tuits (en todos los idiomas) publicados entre el 2 y el 4 de febrero de 2008.
- La estructura de las entradas es la siguiente...

# La colección de documentos

```
{
  "_index": "twitter-archive-2008-feb",
  "_type": "tweet",
  "_id": "668997402",
  "_score": 1.4142135,
  "_source": {
    "created_at": "Sat Feb 02 06:00:19 +0000 2008",
    "id_str": "668997402",
    "in_reply_to_status_id_str": "668959452",
    "in_reply_to_user_id_str": "816952",
    "lang": "en",
    "text": "@RhiannonSL reply to @istarman: Re: Re: Re: Re: Re:
      This loud crazy chick - http://seesmic.com/v/KNdEzV99rU",
    "user_id_str": "9792452",
    "timestamp": 1201932019
  }
}
```

# La colección de documentos

```
{  
  "_index": "twitter-archive-2008-feb",  
  "_type": "tweet",  
  "_id": "668997402",  
  "_score": 1.4142135,  
  "_source": {  
    "created_at": "Sat Feb 02 06:00:19 +0000 2008",  
    "id_str": "668997402",  
    "in_reply_to_status_id_str": "668959452",  
    "in_reply_to_user_id_str": "816952",  
    "lang": "en",  
    "text": "@RhiannonSL reply to @istarman: Re: Re: Re: Re: Re:  
      This loud crazy chick - http://seesmic.com/v/KNdEzV99rU",  
    "user_id_str": "9792452",  
    "timestamp": 1201932019  
  }  
}
```

# La colección de documentos

- Para facilitar el desarrollo del ejercicio puede ser prudente filtrar el archivo por idioma
- Para ello vete a <https://sourceforge.net/projects/unxutils/>
- Descarga el archivo y descomprímelo en tu carpeta de usuario
- Usa `$USUARIO/UnxUtils/usr/local/wbin/grep` para filtrar por idioma (p.ej. `\ "lang\":"\ "en\"` o `\ "lang\":"\ "es\"`)
  - El archivo en inglés debería tener 394.990 tuits
  - El archivo en castellano debería tener 48.622 tuits
- Usa `$USUARIO/UnxUtils/usr/local/wbin/wc` para contar las líneas

# Antes de indexar la colección

- Para indexar la colección es preciso invocar los correspondientes comandos ES
- Podríamos intentar hacerlos desde el front-end pero es mucho más razonable escribir un programa en algún lenguaje de programación para realizar dicha operación.
- El lenguaje de programación es irrelevante pero en las prácticas vamos a utilizar PHP (¡sí, PHP!) – En casa puedes usar Java si lo deseas



# Usando ES con PHP

- Toda la información aquí:  
<https://www.elastic.co/guide/en/elasticsearch/client/php-api/current/index.html>
- **Instalación de PHP:**
  - Vete a <http://windows.php.net/download#php-5.6>
  - Descarga el archivo ZIP (x86 o x64) a tu carpeta de usuario y descomprímelo (en tu casa mejor en C:\PHP\)
- **Configuración del proyecto:**
  - Crea una carpeta para tu proyecto (donde escribirás todo el código PHP que usará el cliente PHP para ES)
  - Descarga en la carpeta el siguiente archivo: <http://getcomposer.org/installer>
  - Ejecuta `C:/php/php.exe installer`
  - Con eso habrás instalado Composer
  - Crea en la carpeta un archivo `composer.json`
  - **CONTINÚA...**

# Usando ES con PHP

- **Configuración del proyecto:**

- CONTINÚA LA TRANSPARENCIA ANTERIOR
- El fichero `composer.json` debe contener lo siguiente:

```
{ "require": { "elasticsearch/elasticsearch": "~5.0" } }
```
- Ejecuta `php composer.phar install --no-dev`
- Ya has descargado todo lo necesario para usar el cliente de PHP para ES en tus scripts PHP
- En los scripts deberías incluir lo siguiente:

```
require 'vendor/autoload.php';  
$client = Elasticsearch\ClientBuilder::create()->build();
```

# Indexando la colección

- Descarga el script `index.php`, estúdialo y ejecútalo.
- Tras la ejecución si visitas la sección *overview* en cerebro debería aparecer algo así...

The screenshot shows the Kibana Overview page for an Elasticsearch cluster. The top navigation bar includes 'overview', 'nodes', 'rest', and 'more'. The main content area displays the following information:

- elasticsearch** (cluster name)
- 1 nodes** (number of nodes)
- 389,961 docs** (total documents)
- 195.83MB** (total size)

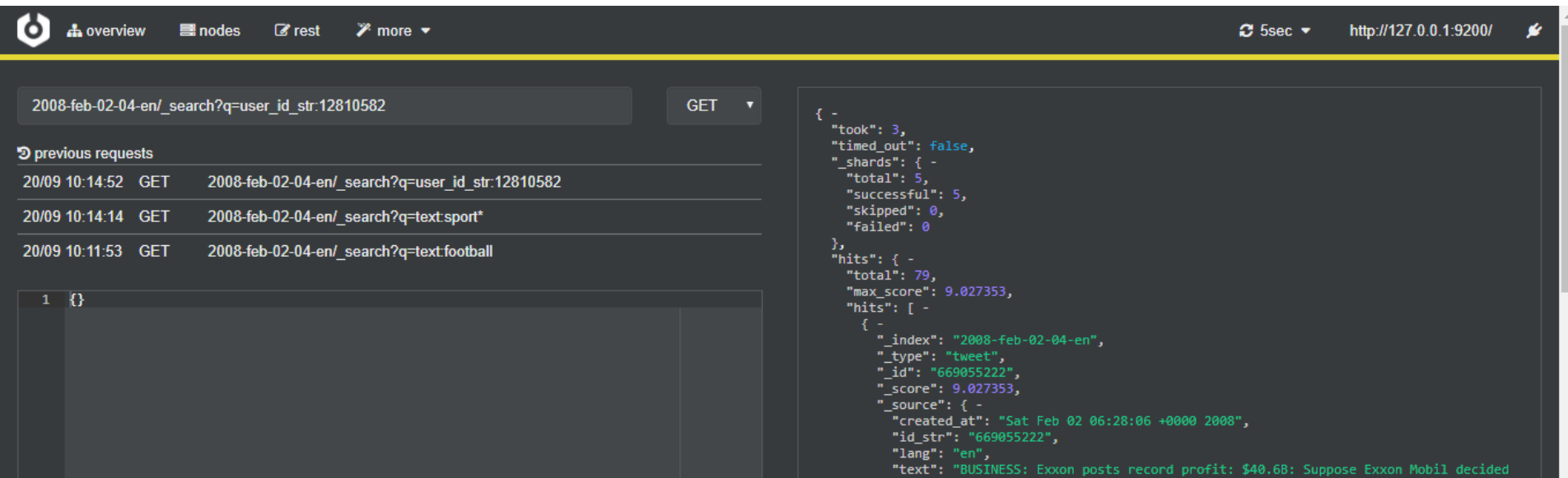
Below this, there are filters for indices and nodes. The index list shows one index: **2008-feb-02-04-en**, with 5 shards and 389,961 documents. A warning indicates **5 unassigned shards**. The index is shown with a star icon and the name **FWgeCaC**. The node list shows one node with the name **127.0.0.1** and a status of **heap**.

Index	Shards	Docs	Size
2008-feb-02-04-en	5 * 2	389,961	195.83MB

Node	Shards	Docs	Size
127.0.0.1	0	0	0

# Lanzando consultas desde cerebro

- Hay que ir a la vista *rest*
- Posibles consultas:
  - `2008-feb-02-04-en/_search?q=text:football`
    - 1571 resultados
  - `2008-feb-02-04-en/_search?q=text:sport*`
    - 1267 resultados
  - `2008-feb-02-04-en/_search?q=user_id_str:12810582`
    - 79 resultados



The screenshot shows the Elasticsearch REST client interface. The top navigation bar includes 'overview', 'nodes', 'rest', and 'more'. The address bar shows 'http://127.0.0.1:9200/'. The main area displays a search query: `2008-feb-02-04-en/_search?q=user_id_str:12810582` with a 'GET' method. Below the query bar, there is a table of 'previous requests' with columns for time, method, and query. The table shows three requests: 20/09 10:14:52 GET 2008-feb-02-04-en/\_search?q=user\_id\_str:12810582, 20/09 10:14:14 GET 2008-feb-02-04-en/\_search?q=text:sport\*, and 20/09 10:11:53 GET 2008-feb-02-04-en/\_search?q=text:football. The response area shows a JSON object: 

```
{}
```

. The right sidebar displays the raw JSON response, which includes metadata like 'took', 'timed\_out', and 'hits'. The 'hits' array contains one result with '\_index': '2008-feb-02-04-en', '\_type': 'tweet', '\_id': '669055222', '\_score': 9.027353, and 'text': 'BUSINESS: Exxon posts record profit: \$40.6B: Suppose Exxon Mobil decided'.

2008-feb-02-04-en/\_search?q=user\_id\_str:12810582 GET

previous requests

time	method	query
20/09 10:14:52	GET	2008-feb-02-04-en/_search?q=user_id_str:12810582
20/09 10:14:14	GET	2008-feb-02-04-en/_search?q=text:sport*
20/09 10:11:53	GET	2008-feb-02-04-en/_search?q=text:football

```
{}
```

```
{
  "took": 3,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 79,
    "max_score": 9.027353,
    "hits": [
      {
        "_index": "2008-feb-02-04-en",
        "_type": "tweet",
        "_id": "669055222",
        "_score": 9.027353,
        "_source": {
          "created_at": "Sat Feb 02 06:28:06 +0000 2008",
          "id_str": "669055222",
          "lang": "en",
          "text": "BUSINESS: Exxon posts record profit: $40.6B: Suppose Exxon Mobil decided"
        }
      }
    ]
  }
}
```

# Consultas con parámetros

- Se pueden especificar parametrizaciones más complejas mediante JSON
- Por ejemplo, forzar el operador **and**

```
{  
  "query": {  
    "match": {  
      "text": {  
        "query": "super bowl tuesday",  
        "operator": "and"  
      }  
    }  
  }  
}  
36 resultados
```

- O solicitar únicamente el número de hits sin documentos resultantes:

```
{  
  "size" : 0,  
  "query" : {  
    "match": {  
      "timestamp": "1202063161"  
    }  
  }  
}  
2 resultados
```

# Ejecutando consultas desde PHP

- El cliente PHP para ES permite especificar los parámetros con la misma estructura que el API REST pero usando arrays asociativos en vez de JSON.
- **Descargar y estudiar el script `query.php`**
- Sobre las consultas de ejemplo realizar modificaciones y comprobar qué resultados ofrece.

# Ejercicio

## Paginación de resultados

- La forma más básica de paginar resultados es usando los parámetros `from` (valor por defecto 0) y `size` (valor por defecto 10)
- El mayor problema es que es **muy ineficiente** y sólo permite llegar a un máximo de **10.000 resultados**
- Estudiar la siguiente documentación:
  - <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-from-size.html>
  - <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-scroll.html>
  - <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-search-after.html>
  - <https://www.elastic.co/guide/en/elasticsearch/client/php-api/current/search-operations.html> (buscar Scrolling)
- Implementar un script que permita obtener de manera eficiente documentos para consultas que produzcan más de 10.000 resultados (p.ej. `/_search?q=text:the`)

# Un tutorial interesante

- A Practical Introduction to Elasticsearch
  - <https://www.elastic.co/blog/a-practical-introduction-to-elasticsearch>
- Especialmente interesante la parte de agregaciones (aunque tal vez no sea particularmente útil con una colección tan poco estructurada como la de tuits...)



# Posibles prácticas con ES

- Usando la colección del tutorial:
  - Generar series temporales para términos introducidos por un usuario (similar a Google Trends)
    - Ver <https://www.quora.com/What-are-the-best-JavaScript-libraries-for-time-series-graphs> para elegir una biblioteca para dibujo de series
  - Encontrar los términos (no vacíos) más relacionados con un término dado (relacionado con expansión de consultas)
  - Combinado con Neo4J:
    - Obtener los grafos de interacción entre usuarios hablando de la Super Bowl
    - Obtener grafos de retuits
    - Opcionalmente: visualizar los grafos de interacción de las dos hinchadas usando Gephi (<https://gephi.org/>)
  - Más opciones: a discutir con Dani
- Usando otras colecciones:
  - Colecciones de tuits:
    - Disponibles aquí: <https://archive.org/details/twitterstream>
    - Generadas desde aquí: <http://danigayo.info/twitter-toddler/> (estudiante debería indicar keywords para que Dani genere un *dump*)
    - Prácticas similares a las propuestas para el tutorial o a discutir con Dani
  - Otras colecciones: a discutir con Dani