



RECUPERACIÓN DE LA INFORMACIÓN

Solr – Elasticsearch – Logstash - Kibana

Asignatura:

Introducción A La Recuperación De La Información

Profesor:

Eric Jeltsch Figueroa

Alumnos:

Cristian Araya – Rodrigo Ayala – Daniel Barraza
Luis Cuello – Felipe Fernández – Cristhian García - Ariel Perez

Índice

Introducción	2
Solr	3
SolrCloud	3
Indexación de Archivos	4
Consola de Comandos	4
Interfaz de Usuario	6
SolrJ (Java API)	8
Base de Datos	10
Búsqueda	11
Búsquedas Geoespaciales	15
ElasticSearch	25
Explorando el Cluster	25
Indexación y Búsqueda	25
Explorando el Contenido	30
Relevancia (Score)	32
Teoría detrás de la Relevancia (Scoring)	34
Logstash y Kibana	37
Indexar y visualizar logs por cmd	38
Indexar y visualizar logs obtenidos desde un archivo .csv a través de logstash	42
Indexar y visualizar logs obtenidos desde un .csv generado desde apache jmeter al realizar una petición http a un sitio web	46
Conclusión	51

1. Introducción

Solr posee un conjunto de herramientas para realizar búsquedas en documentos indexados (construido sobre Lucene). Estas herramientas son las características centrales, con ellas se pueden realizar búsquedas personalizadas y a la medida. Las herramientas que se explorarán en este paper son; SolrCloud, indexación de archivos xml, json, y csv, visualización de resultados en formato json a través de la consola de comandos, interfaz gráfica de usuario (adminUI), programa externo y base de datos, y finalmente variantes para realizar búsquedas.

Por otra parte Elasticsearch, Logstash y Kibana, proveen una suite más completa para la indexación y búsqueda de archivos, desde el resultado de la relevancia de lo que se está buscando, hasta apoyos visuales con gráficas de los datos indexados. Si bien ambas herramientas, Solr y Elasticsearch, utilizan Lucene como base, en este informe se verá que este último resulta ser una opción más completa, aún así ambas herramientas otorgan funcionalidades bien definidas y trabajadas.

2. Solr

2.1. SolrCloud

Apache Solr incluye la habilidad de preparar un cluster de servidores Solr que combinan tolerancia a fallos y alta disponibilidad, a esto Solr lo llamo SolrCloud. Estas capacidades proveen indexaciones y búsquedas distribuidas.

La indexación y búsqueda en SolrCloud es flexible al no usar un nodo maestro (se ocupa más de un nodo). Las solicitudes pueden ser mandadas a cualquier servidor. Mediante Zookeeper se decide, de forma interna, cual servidor necesita manejar las solicitudes. Zookeeper es otra herramienta de Apache la cual se define como:

ZooKeeper es un servicio centralizado para mantener información de configuración, nombrar, proporcionar sincronización distribuida y proporcionar servicios de grupo. Todos estos tipos de servicios son usados desde o por otras aplicaciones distribuidas.”

Aunque suene complicado, básicamente SolrCloud sirve cuando se necesita trabajar con varios servidores. Las funcionalidades para el usuario siguen siendo las mismas, pero de forma interna Solr trabaja de forma muy diferente a cuando es ejecutado de forma normal, un ejemplo de esto y que se destaca a simple vista son las 'Colecciones', mientras que en una ejecución de Solr normal, se habla de core (núcleo) en SolrCloud se habla de colecciones, con un core por nodo creado (ver figura 1).

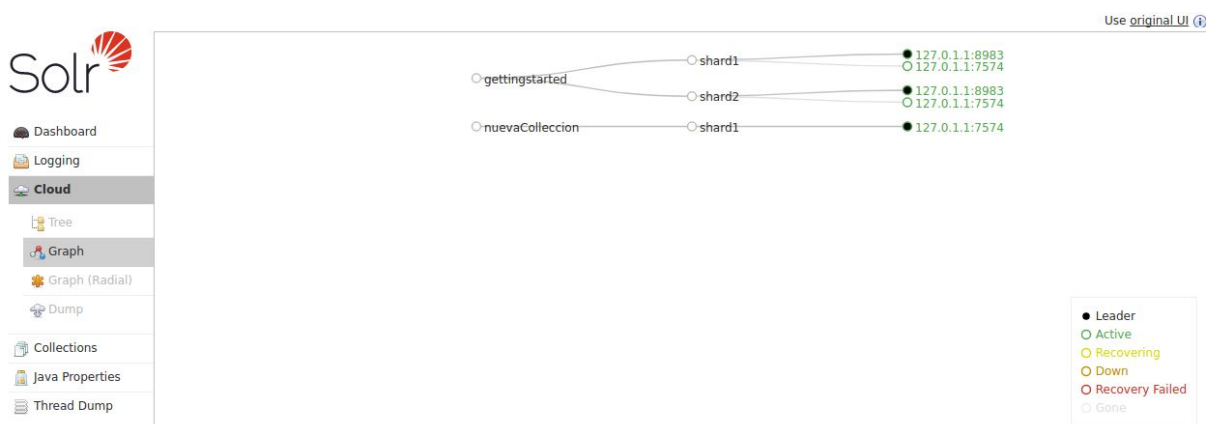


Figura 1. Gráfico SolrCloud

En las dos secciones siguientes se utilizará SolrCloud para indexar por comandos y por la interfaz AdminUI, de esta forma se mostrará que para el usuario las funcionalidades son las mismas con solo pequeñas variaciones.

2.2. Indexación de Archivos

Un índice de Solr puede aceptar datos desde varias fuentes, incluyendo archivos XML, archivos CSV (valores separados por coma), datos extraídos desde tablas en una base de datos, y archivos con formatos comunes como Microsoft Word o PDF. Siempre existiría una estructura de datos para alimentar al índice de Solr (independiente del método usado para hacerlo), esto es, un documento conteniendo múltiples campos, cada uno con un nombre y contenido, el cual puede estar vacío, con uno de los campos usualmente designado como un campo de ID único (análogo a una clave primaria en una base de datos).

```
"id":"SP2514N",
"name":["Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133"],
"manu":["Samsung Electronics Co. Ltd."],
"manu_id_s":"samsung",
"cat":["electronics",
      "hard drive"],
"features":["7200RPM, 8MB cache, IDE Ultra ATA-133",
           "NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor"],
"price":[92.0],
"popularity":[6],
"inStock":[true],
"manufacturedate_dt":"2006-02-13T15:26:37Z",
"store":["35.0752,-97.032"],
"_version_":1566458552631951360},
```

Figura 2. Estructura de un Documento

2.2.1. Consola de Comandos

La forma más directa de realizar indexaciones y búsqueda a Solr es usando la consola de comandos (o terminal en Ubuntu). Para realizarlas primero se debe iniciar Solr con el siguiente comando (ubicado en la carpeta raíz de Solr):

```
bin/solr start -e cloud -noprompt
```

En este comando estamos iniciando a Solr, especificando la carpeta que se utilizara, en este caso 'cloud', mientras que el parámetro 'noprompt' le indica a Solr que ocupe la configuración por defecto. Si el parámetro noprompt no se hubiera utilizado en el comando, Solr pediría al usuario los puertos que los nodos ocuparían y la colección en la que se trabajara. La carpeta cloud es un ejemplo que viene con Solr, los puertos que ocupa son el 8983 y 7574 para el nodo1 y nodo2 respectivamente, y la colección gettingstarted para trabajar. El próximo comando en la indexación es:

```
bin/post -c gettingstarted docs/
```

Este comando le indica a Solr el nombre de la colección en la cual se indexara y la carpeta de los documentos a indexar.

Con este simple comando que ejecuta la funcionalidad 'post' (bin/post) se realizan todas las indexaciones a Solr. La figuras 3, 4, y 5 muestra cómo se indexan archivos en formato xml, csv, y json con los respectivos mensajes de respuestas de Solr. Notar que en cada mensaje Solr detalla el .jar que fue utilizado junto con la clase usada (SimplePostTool).

```
n/post -c gettingstarted example/exampledocs/*.xml
java -classpath /home/sefirot/Documents/College/information_retrival/solr/dist/s
olr-core-6.5.1.jar -Dauto=yes -Dc=gettingstarted -Ddata=files org.apache.solr.ut
il.SimplePostTool example/exampledocs/gb18030-example.xml example/exampledocs/hd
.xml example/exampledocs/ipod_other.xml example/exampledocs/ipod_video.xml exam
ple/exampledocs/manufacturers.xml example/exampledocs/mem.xml example/exampledocs
/money.xml example/exampledocs/monitor2.xml example/exampledocs/monitor.xml exam
ple/exampledocs/mp500.xml example/exampledocs/sd500.xml example/exampledocs/solr
.xml example/exampledocs/utf8-example.xml example/exampledocs/vidcard.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/gettingstarted/update...
Entering auto mode. File endings considered are xml,json,jsonl,csv,pdf,doc,docx,
ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,txt,log
POSTing file gb18030-example.xml (application/xml) to [base]
POSTing file hd.xml (application/xml) to [base]
POSTing file ipod_other.xml (application/xml) to [base]
POSTing file ipod_video.xml (application/xml) to [base]
POSTing file manufacturers.xml (application/xml) to [base]
POSTing file mem.xml (application/xml) to [base]
POSTing file money.xml (application/xml) to [base]
POSTing file monitor2.xml (application/xml) to [base]
POSTing file monitor.xml (application/xml) to [base]
POSTing file mp500.xml (application/xml) to [base]
POSTing file sd500.xml (application/xml) to [base]
POSTing file solr.xml (application/xml) to [base]
POSTing file utf8-example.xml (application/xml) to [base]
POSTing file vidcard.xml (application/xml) to [base]
14 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/gettingstarted/updat
e...
Time spent: 0:00:09.509
```

Figura 3. Indexación Archivos xml

```
bin/post -c gettingstarted example/exampledocs/books.csv
java -classpath /home/sefirot/Documents/College/information_retrival/solr/dist
/solr-core-6.5.1.jar -Dauto=yes -Dc=gettingstarted -Ddata=files org.apache.sol
r.util.SimplePostTool example/exampledocs/books.csv
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/gettingstarted/update..
.
Entering auto mode. File endings considered are xml,json,jsonl,csv,pdf,doc,doc
x,ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,txt,log
POSTing file books.csv (text/csv) to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/gettingstarted/upd
ate...
Time spent: 0:00:01.693
```

Figura 4. Indexación Archivos csv


```

bin/post -c gettingstarted example/exampledocs/books.json
java -classpath /home/sefirot/Documents/College/information_retrival/solr/dist
/solr-core-6.5.1.jar -Dauto=yes -Dc=gettingstarted -Ddata=files org.apache.sol
r.util.SimplePostTool example/exampledocs/books.json
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/gettingstarted/update..
.
Entering auto mode. File endings considered are xml,json,jsonl,csv,pdf,doc,doc
x,ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,txt,log
POSTing file books.json (application/json) to [base]/json/docs
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/gettingstarted/upd
ate...
Time spent: 0:00:02.625

```

Figura 5. Indexación Archivos json

2.3. Interfaz de Usuario

Otra forma de indexar archivos es a través de la interfaz de usuario que ofrece Solr. Este método ofrece un entorno más familiar para el usuario, ofreciendo las mismas posibilidades que la consola de comandos.

En la página del administrador de Solr se debe seleccionar la colección o el core en el cual se indexara (ver figura 6), una vez seleccionada, se mostrará la pestaña 'Documents' en ella se podrán ingresar los documentos que se deseen indexar según el formato que se especifique (csv, json, xml). La gran diferencia con la consola de comandos que ofrece la interfaz de usuario es la posibilidad de construir el documento mediante la opción 'Document Builder' (ver figura 7). Esta opción permite seleccionar el campo que se quiere agregar al documento (se pueden agregar múltiples campos) mientras que Solr construye el documento, en formato json, en el cuadro de texto 'Document(s)'.

Solr

Request-Handler (qt)
/update

Document Type
JSON

Document(s)

```
{
  "id": "006",
  "name": "Carla",
  "lastname": "Jimenez",
  "age": "22"
},
{
  "id": "007",
  "name": "Nicolas",
  "lastname": "Perez",
}
```

Commit Within
1000

Overwrite
true

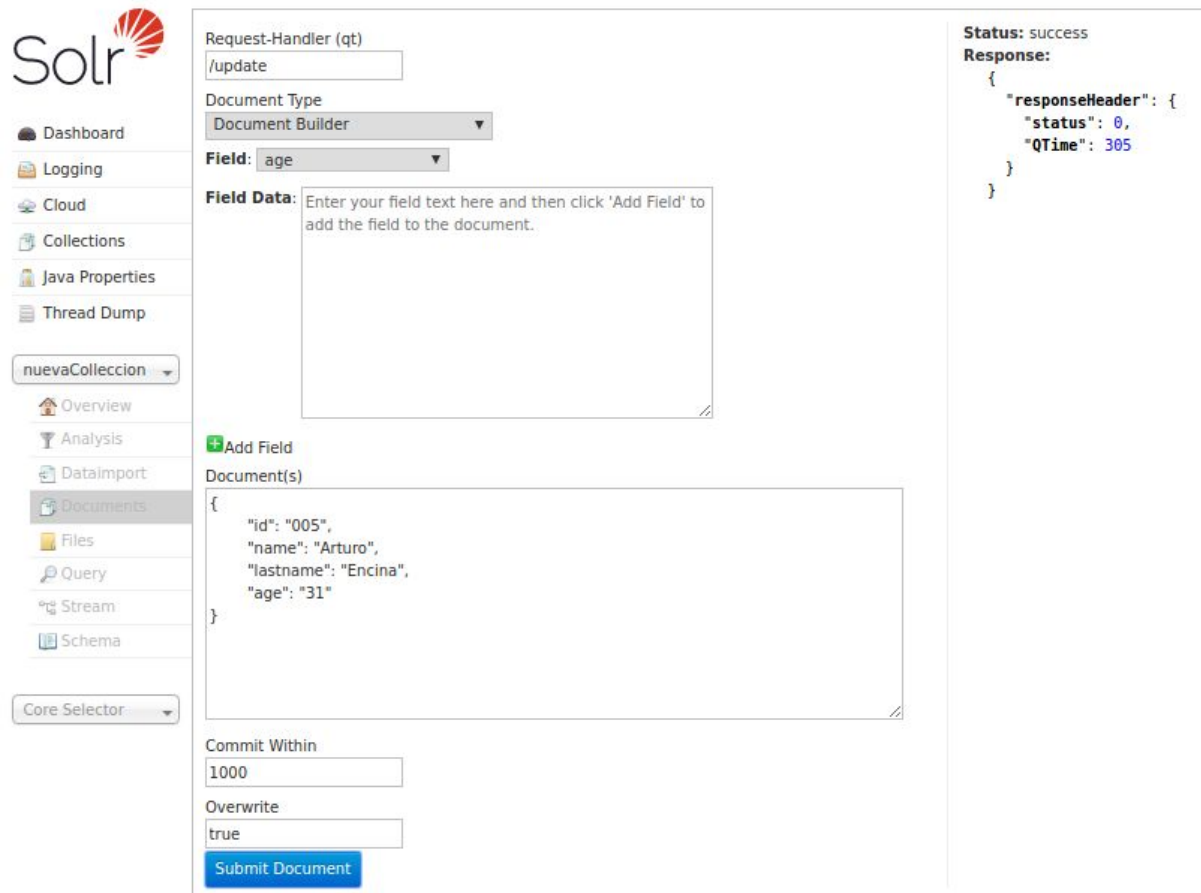
Boost
1.0

Submit Document

Status: success
Response:

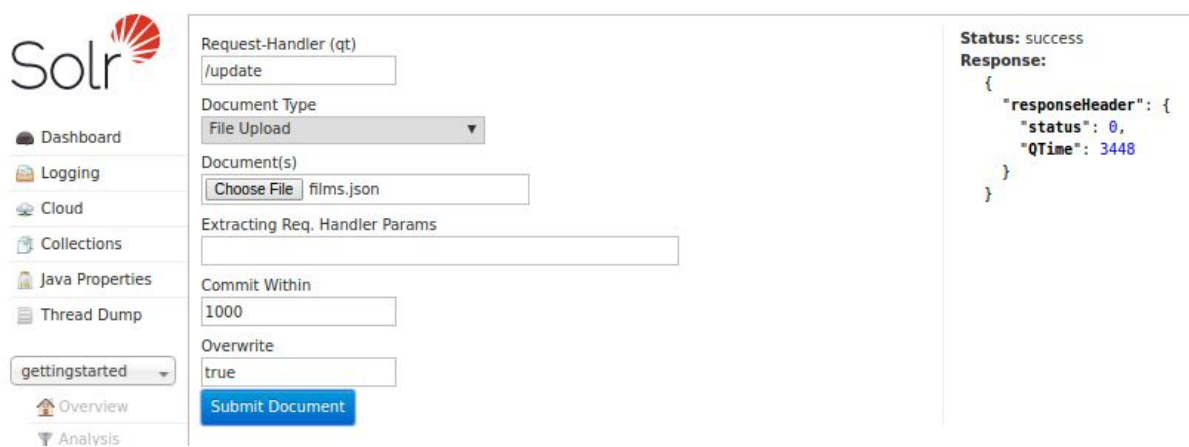
```
{
  "responseHeader": {
    "status": 0,
    "QTime": 20
  }
}
```

Figura 6. UI Indexación Archivos json



The image shows the Solr Document Builder interface. On the left is a sidebar with navigation links: Dashboard, Logging, Cloud, Collections, Java Properties, Thread Dump, nuevaColeccion (selected), Overview, Analysis, DataImport, Documents (selected), Files, Query, Stream, and Schema. Below these is a 'Core Selector' dropdown. The main area is titled 'Request-Handler (qt)' with a '/update' input. Below this is a 'Document Type' dropdown set to 'Document Builder'. A 'Field:' dropdown is set to 'age'. The 'Field Data:' section contains a text area with the instruction: 'Enter your field text here and then click 'Add Field' to add the field to the document.' Below this is a green '+Add Field' button. The 'Document(s)' section shows a JSON object: { "id": "005", "name": "Arturo", "lastname": "Encina", "age": "31" }. At the bottom are 'Commit Within' (1000), 'Overwrite' (true), and a 'Submit Document' button. On the right, the 'Status: success' and 'Response:' are shown, with a JSON response: { "responseHeader": { "status": 0, "QTime": 305 } }.

Figura 7. Document Builder



The image shows the Solr File Upload interface. The sidebar is similar to Figure 6, but 'Documents' is not selected. The main area is titled 'Request-Handler (qt)' with a '/update' input. Below this is a 'Document Type' dropdown set to 'File Upload'. The 'Document(s)' section has a 'Choose File' button and the filename 'films.json'. Below this is an 'Extracting Req. Handler Params' input field. At the bottom are 'Commit Within' (1000), 'Overwrite' (true), and a 'Submit Document' button. On the right, the 'Status: success' and 'Response:' are shown, with a JSON response: { "responseHeader": { "status": 0, "QTime": 3448 } }.

Figura 8. Documentos a traves de Archivo

2.4. SolrJ (Java API)

SolrJ es una API que hace m´as f´acil la comunicaci3n de las aplicaciones Java con Solr. SolrJ esconde los detalles de conexi3n con Solr y permite a las aplicaciones interactuar con Solr usando m3todos simples de alto nivel. El centro de SolrJ es el paquete `org.apache.solr.client.solrj`, con este paquete se pueden crear clientes para realizar indexaciones o consultas a Solr.

En la figura Indexaci3n con SolrJ se puede apreciar los pasos para construir una aplicaci3n que indexe datos a Solr. Esto se realiza especificando la url del core en el cual se desea indexar, o el core de la colecci3n en caso de estar usando SolrCloud, para crear el cliente (SolrClient) con el cual se agregara el nuevo documento. Los documentos se crean utilizando la clase `SolrInputDocument`, el m3todo `addField` es el encargado de agregar campos al documentos, una vez que el documento est´a listo para la indexaci3n, el m3todo `add` del cliente de Solr lo agregara mientras que el m3todo `commit` confirma los cambios, de esta forma en pocos pasos se realiza la indexaci3n. Para realizar b3squedas en Solr tambi3n se necesitan pocos pasos. Los conceptos son los mismos de la indexaci3n, la gran diferencia recae en la clase `SolrQuery`, la cual es la encargada de preparar la consulta que se realizar´a. El cliente de Solr es el que ejecuta la consulta, devolviendo la respuesta de la cual se obtienen los resultados como una lista de documentos de Solr, sobre los cuales se puede iterar. Como se puede ver en la figura B´usqueda con SolrJ, las propiedades disponibles del resultado devuelto son las mismas que se presentan en las b3squedas a trav3s del AdminUI y la l3nea de comandos (ver figura 11).

```
import java.io.IOException;

import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrServerException;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.apache.solr.common.SolrInputDocument;

public class AgregarDocumento {
    public static void main(String args[]) throws Exception {
        //Preparando el cliente de Solr.
        String urlString = "http://localhost:8983/solr/gettingstarted";
        SolrClient solr = new HttpSolrClient.Builder(urlString).build();

        //Preparando el documento de Solr.
        SolrInputDocument doc = new SolrInputDocument();

        //Agregando campos al documento.
        doc.addField("id", "004");
        doc.addField("name", "Juana");
        doc.addField("lastname", "Nieves");
        doc.addField("age", "20");

        //Agregando el documento a Solr.
        solr.add(doc);

        //Guardando los cambios.
        solr.commit();
        System.out.println("Documentos agregados");
    }
}
```

Figura 9. Indexación con SolrJ

```

import java.io.IOException;

import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.SolrServerException;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.apache.solr.common.SolrDocumentList;

public class BuscarDocumento {

    public static void main(String[] args) throws SolrServerException, IOException {
        String urlString = "http://localhost:8983/solr/gettingstarted";
        SolrClient solr = new HttpSolrClient.Builder(urlString).build();

        SolrQuery query = new SolrQuery();
        query.set("q", "*:*");
        QueryResponse response = solr.query(query);

        SolrDocumentList docList = response.getResults();
        System.out.println("NumFound" + docList.getNumFound());

        for (SolrDocument doc : docList) {
            System.out.println("Id: " + doc.getFieldValue("id"));
            System.out.println("Name: " + doc.getFieldValue("name"));
            System.out.println("Lastname: " + doc.getFieldValue("lastname"));
            System.out.println("Age: " + doc.getFieldValue("age"));
        }
    }
}

```

Figura 10. Búsqueda con SolrJ

```

NumFound4
Id: 001
Name: [Rodolfo]
Lastname: [Urrutia]
Age: [18]
Id: 002
Name: [Barbara]
Lastname: [Lopez]
Age: [21]
Id: 003
Name: [Sofia]
Lastname: [Tapia]
Age: [29]
Id: 004
Name: [Hector]
Lastname: [Vargas]
Age: [28]

```

Figura 11. Resultado Búsqueda con SolrJ

2.5. Base de Datos

Como se mencionó anteriormente, Solr puede ser alimentado a través de base de datos. En esta ocasión, se explicará cómo indexar datos desde una base de datos Mysql a un core de Solr (no se utilizara SolrCloud).

Para realizar la indexación se necesitan de tres archivos; solrconfig.xml, managed-schema (o también llamado schema.xml), y db-data-config.xml, con estos tres archivos se especificarán las configuraciones necesarias para utilizar la funcionalidad DataImport del AdminUI que es la encargada de realizar la indexación. El primer paso a realizar es la configuración del archivo solrconfig.xml, en él se debe especificar la o las librerías que se ocuparan para la importación de los datos (figura 12), además de especificar la clase y el archivo de configuración para los datos que utilizará el requesthandler (figura 13).

```
<lib dir="${solr.install.dir:../../../../..}/dist/" regex="solr-dataimporthandler-.*\.jar" />
```

Figura 12. Librería para el DataImport

```
<requestHandler name="/dataimport" class="solr.DataImportHandler">
  <lst name="defaults">
    <str name="config">db-data-config.xml</str>
  </lst>
</requestHandler>
```

Figura 13. RequestHandler para el DataImport

En el archivo solrconfig.xml se configuran características importantes de Solr, entre ellas se encuentran los requesthandlers (manipuladores de solicitudes) los cuales procesan las solicitudes a Solr, tales como:

- Solicitudes para agregar documentos al índice.
- Solicitudes para retornar resultados de consultas.

El segundo paso es modificar el archivo managed-schema. Solr almacena los detalles acerca de los campos y tipos de campos de los documentos en este archivo. En la figura Campos del Esquema se puede apreciar que se ha agregado un campo personalizado al esquema (el campo área), estos campos son los utilizados en las búsquedas e indexaciones que se realizan.

Por último queda modificar el archivo db-data-config.xml, el cual fue especificado en el requesthandler en el archivo solrconfig.xml. En este archivo se define los datos de conexión a la base de datos de la cual se importarán los datos, url, usuario y contraseña, como las columnas que se agregan al documento a indexar (ver figura 14). Cabe destacar que Solr necesita del driver de java del motor de base de datos que se esté utilizando, en este caso

se necesita del driver de Mysql (mysql-connector-java-5.1.32-bin.jar), el cual se debe situar en la carpeta "lib" dentro de la carpeta del core que se esté utilizando. Con estas configuraciones se puede utilizar el DataImport desde el AdminUI para indexar documentos desde bases de datos (ver figura 15).

```
<field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />
<field name="sku" type="text_en_splitting_tight" indexed="true" stored="true" omitNorms="true"/>
<field name="name" type="text_general" indexed="true" stored="true"/>
<field name="manu" type="text_general" indexed="true" stored="true" omitNorms="true"/>
<field name="area" type="text_general" indexed="true" stored="true" multiValued="true"/>
<field name="cat" type="string" indexed="true" stored="true" multiValued="true"/>
<field name="features" type="text_general" indexed="true" stored="true" multiValued="true"/>
<field name="includes" type="text_general" indexed="true" stored="true" termVectors="true" termPositions="true" termOffsets="true" />
```

Figura 14. Campos del Esquema

```
<dataConfig>
  <dataSource driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost:3306/empleados" user="root" password="1234" />
  <document>
    <entity name="empleado" query="select id, name, area from empleado;">
      <field column="name" name="name" />
      <field column="area" name="area" />
    </entity>
  </document>
</dataConfig>
```

Figura 15. db-data-config.xml

```
mysql> select * from empleado;
+----+-----+-----+
| id | name       | area   |
+----+-----+-----+
| 1  | Juan Nieves | Soporte |
| 2  | Juana Nieves | Recursos |
| 3  | Matias Rojas | Soporte |
+----+-----+-----+
3 rows in set (0,00 sec)
```

Figura 16. BD Empleados

2.6. Búsqueda

Para realizar búsquedas en Solr existe una sintaxis específica. Esta sintaxis permite construir consultas personalizadas para realizarlas. La búsqueda más simple es la que viene por defecto en la interfaz de usuario de Solr "*,*". Esta consulta le indica a Solr que queremos buscar todos los archivos sin filtro alguno (aunque en realidad existe el filtro del límite de documentos a devolver). Por ejemplo el comando:

```
curl "http://localhost:8983/solr/gettingstarted/select?indent=on&q=*,*&wt=json"
```

en la terminal le indicará a Solr que busque en la colección gettingstarted todos los archivos (parámetro q valor "*,*") y que la respuesta sea en formato json (parámetro wt valor json).

Para realizar búsquedas por un único término se define el valor del parámetro q con el valor a buscar, por ejemplo


```
curl "http://localhost:8983/solr/gettingstarted/select?wt=json&indent=true&q=foundation"
```

De esta forma se buscarán los documentos que contengan el valor. Si se quiere buscar por frases el espacio debe ser reemplazado por %20 como en el siguiente ejemplo:

```
curl "http://localhost:8983/solr/gettingstarted/select?indent=on&q=CAS %20latency&wt=json"
```

Otra de las características de la búsqueda es el 'facet' el cual permite agrupar el resultado en subsets entregando el conteo de cada subset. El siguiente comando es un ejemplo de esto:

```
curl 'http://localhost:8983/solr/gettingstarted/select?wt=json&indent=true&q=*&rows=0'&facet=true&facet.field=manufacturer'
```

En la figura Búsqueda con Facet se puede observar el resultado de la consulta. Como se puede ver el facet se realiza por el id del fabricante, y el resultado muestra el número de documentos por fabricante.

Otra gran característica es el pivot en los facets, también conocido como árboles de decisión, el cual permite a dos o más campos ser enlazados para todas las posibles combinaciones. El siguiente comando realiza una búsqueda en la cual el resultado especifica si los productos en la categoría libros están o no en stock (ver figura 20).

```
curl 'http://localhost:8983/solr/gettingstarted/select?q=*&rows=0&wt=json&indent=on'&facet=on&facet.pivot=cat,inStock'
```

Solr posee muchas más características en las búsquedas, la sintaxis que define permite realizar avanzadas consultas, lo que lo transforma (junto con su flexibilidad en los formatos) en una de las funcionalidades más poderosas.

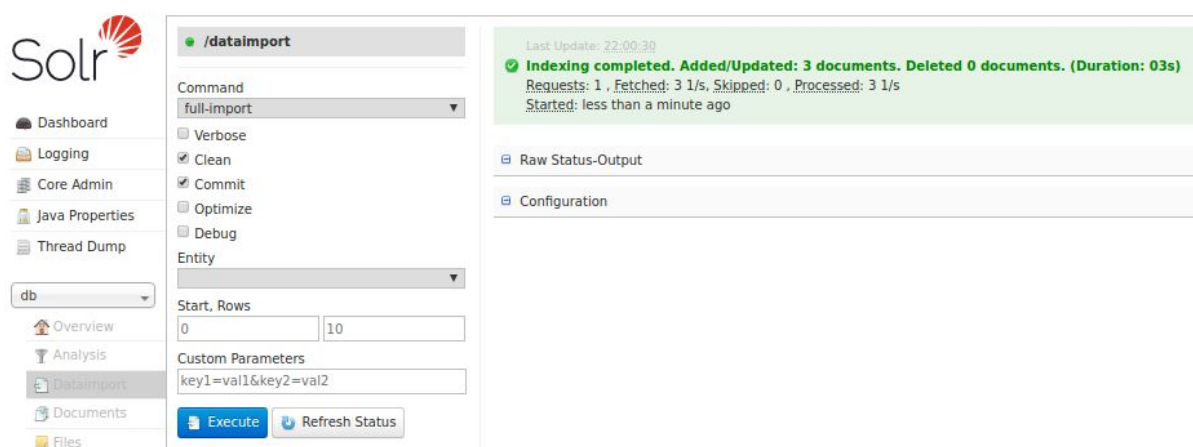


Figura 17. DataImport Indexación

Request-Handler (qt)

/select

— common —

q

:

fq

sort

start, rows

010

fl

df

Raw Query Parameters

key1=val1&key2=val2

wt

json

☒ indent

☐ debugQuery

☐ dismax

☐ edismax

☐ hl

☐ facet

☐ spatial

☐ spellcheck

Execute Query

http://localhost:8983/solr/db/select?indent=on&q=*

```

{
  "responseHeader":{
    "status":0,
    "QTime":3,
    "params":{
      "q":":*",
      "indent":"on",
      "wt":"json",
      " _ ":"1493859679372"}},
  "response":{"numFound":3,"start":0,"docs":[
    {
      "area":["Soporte"],
      "name":"Juan Nieves",
      "id":"1",
      "_version_":1566425354106568704},
    {
      "area":["Recursos"],
      "name":"Juana Nieves",
      "id":"2",
      "_version_":1566425354244980736},
    {
      "area":["Soporte"],
      "name":"Matias Rojas",
      "id":"3",
      "_version_":1566425354247077888}]
  }}

```

Figura 18. Búsqueda de los documentos indexados

```

"response":{"numFound":1149,"start":0,"maxScore":1.0,"docs":[]
},
"facet_counts":{
  "facet_queries":{},
  "facet_fields":{
    "manu_id_s":[
      "corsair",3,
      "belkin",2,
      "canon",2,
      "apple",1,
      "asus",1,
      "ati",1,
      "boa",1,
      "dell",1,
      "eu",1,
      "maxtor",1,
      "nor",1,
      "samsung",1,
      "uk",1,
      "viewsonic",1]],
    "facet_ranges":{},
    "facet_intervals":{},
    "facet_heatmaps":{}}}}

```

Figura 19. Búsqueda con Facet

```

"facet_pivot":{
  "cat,inStock":[{
    "field":"cat",
    "value":"book",
    "count":14,
    "pivot":[{
      "field":"inStock",
      "value":true,
      "count":12},
      {
        "field":"inStock",
        "value":false,
        "count":2}]]},
    {
      "field":"cat",
      "value":"electronics",
      "count":12,
      "pivot":[{
        "field":"inStock",
        "value":true,
        "count":8},
        {
          "field":"inStock",
          "value":false,
          "count":4}]]},
    {
      "field":"cat",
      "value":"currency",
      "count":4,
      "pivot":[{
        "field":"inStock",
        "value":true,
        "count":4}]]},
    {

```

Figura 20. Búsqueda con Pivot Facet

2.7. Búsquedas Geoespaciales

Solr se caracteriza por su capacidad de realizar búsquedas basadas en ubicación. Esto se implementa más comúnmente mediante la indexación de un campo, donde cada documento que contiene un punto geográfico (una latitud y longitud), a partir de esto, Solr (dependiendo de las consultas que se realicen) permite filtrar documentos que no caen dentro de un radio especificado en relación a un punto.

Solr contiene dos implementaciones de búsqueda geoespacial primarias:

Búsqueda geoespacial de un punto filtrando utilizando el radio:

Admite la búsqueda de radio basada en un solo par de latitud / longitud y también devolver la distancia del punto buscado en relación a los demás documentos, filtrando valores que están demasiado separados.

Búsqueda geoespacial filtrando por uno o más puntos utilizando el radio y otras formas como polígonos:

Este tipo de búsqueda funciona indexando formas como una serie de cuadros de coordenadas de cuadrícula y luego formando una consulta a través de los cuadros de cuadrícula indexados para buscar rápidamente en un gran número de documentos sin tener que calcular distancias.

- ***Búsqueda geoespacial de un punto***

Usualmente esta búsqueda se realiza utilizando las coordenadas de ubicación latitud y longitud. El filtrado basado en un radio circular (gfilt) o basado en un cuadrado (bbox) con lados iguales al diámetro del radio circular, se dice que es más rápido para calcular este último.

Para llevar a cabo la búsqueda debemos tener creado nuestro core, se crea de la misma forma que en los casos anteriores.

```
C:\Users\Cristhian\Desktop\solr-6.5.0\bin>solr create -c geospatial
Copying configuration to new core instance directory:
C:\Users\Cristhian\Desktop\solr-6.5.0\server\solr\geospatial
Creating new core 'geospatial' using command:
http://localhost:8983/solr/admin/cores?action=CREATE&name=geospatial&instanceDir=geospatial
{
  "responseHeader":{
    "status":0,
    "QTime":2573},
  "core":"geospatial"}
```

Definir los campos de ubicación

Se debe modificar schema.xml para incluir un tipo de campo que contenga nuestra ubicación geográfica:

```
<fieldType name="location" class="solr.LatLonType" subFieldSuffix="_coordinate"/>
```

La clase LatLonType, define el campo ubicación de forma que recibe coordenadas en forma de par latitud, longitud y las separa para tratarlas en campos separados. Para mapear la latitud y la longitud en dos campos separados estos dos campos también necesitarán existir en schema.xml.

```
<dynamicField name="*_coordinate" type="tdouble" indexed="true" stored="false" />
```

Con el campo de ubicación definido y con un campo dinámico puesto en su lugar para que el campo de ubicación trace las coordenadas de latitud y longitud en forma separada, todo lo que queda es enviar (indexar) documentos al motor de búsqueda.

Ejemplo presentado en el libro para indexar:

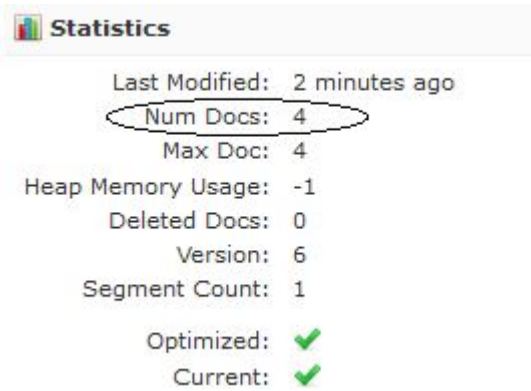
```
<add>
  <doc>
    <field name="id">1</field>
    <field name="location">33.748,-84.391</field>
    <field name="city">Atlanta, GA</field>
  </doc>
  <doc>
    <field name="id">2</field>
    <field name="location">40.715,-74.007</field>
    <field name="city">New York, NY</field>
  </doc>
  <doc>
    <field name="id">3</field>
    <field name="location">37.775,-122.419</field>
    <field name="city">San Francisco, CA</field>
  </doc>
  <doc>
    <field name="id">4</field>
    <field name="location">37.445,-122.161</field>
    <field name="city">Palo Alto, CA</field>
  </doc>
</add>
```

Creé un archivo con esta información con extensión xml y lo ubiqué en la carpeta de ../examples/exampledocs

Indexando...

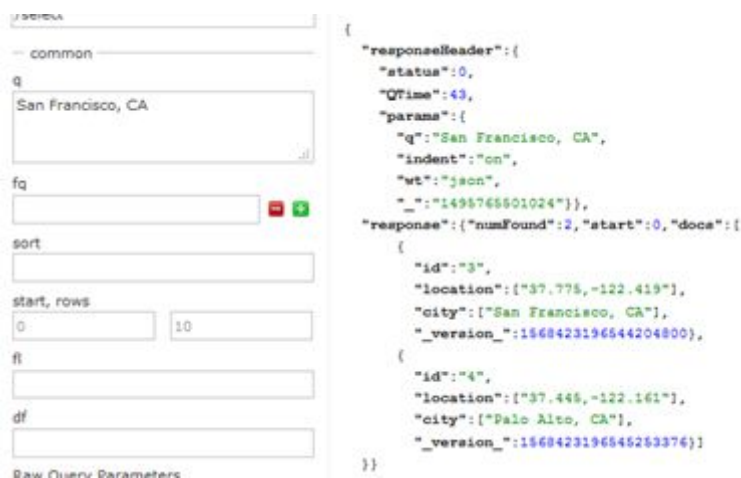
```
C:\Users\Cristhian\Desktop\solr-6.5.0\example\exampledocs>java -Dc=geospatial -jar post.jar geospatial.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/geospatial/update using content-type application/xml...
POSTing file geospatial.xml to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/geospatial/update...
Time spent: 0:00:00.741
```

Para verificar que la indexación fue correcta verificamos en Solr Admin la cantidad de documentos ingresados.



Una vez que nos aseguramos que los documentos están indexados correctamente, es posible buscar en las ubicaciones de varias maneras.

Podemos realizar una simple consulta en modo texto como ciudad: "San Francisco, CA".



El problema con este enfoque es que no será capaz de encontrar documentos que están cerca, no nos proporciona mucha información...

Filtros geográficos y cuadros delimitadores

Una consulta más elaborada sería consultar los documentos dentro de una distancia específica de una ubicación determinada.

Solr contiene un analizador de consultas especial llamado *geofilt* que toma una coordenada de latitud / longitud, un campo de ubicación y una distancia máxima (en km) y sólo coincide con los documentos que caen dentro del área geográfica especificada.

Request-Handler (qt)
/select
-- common
+,*
fq
{!geofilt sfield=location pt=37.77
sort
start, rows
0 10
R
df

http://localhost:5983/solr/search_gspatial/select?fq={!geofilt sfield=location pt=37.77,-122.419 d=20}&indent=on&q=*:*&wt=json
{
 "responseHeader":{
 "status":500,
 "QTime":2,
 "params":{
 "q":"*:*",
 "indent":"on",
 "fq":"{!geofilt sfield=location pt=37.77,-122.419 d=20}",
 "wt":"json",
 "s":"1498772072795"}},
 "error":{
 "metadata":{
 "error-class","org.apache.solr.common.SolrException",
 "root-error-class","org.apache.solr.common.SolrException"},
 "msg":"The field location does not support spatial filtering",
 "trace":["org.apache.solr.common.SolrException: The field location does not support spatial filtering\r\n\tat org.apache.s"]
 "code":500}}

```

{
  "responseHeader":{
    "status":500,
    "QTime":11,
    "params":{
      "q":"*:*",
      "indent":"on",
      "fq":"{!geofilt sfield=location pt=37.77,-122.419 d=20}",
      "wt":"json"}},
  "error":{
    "metadata":{
      "error-class","org.apache.solr.common.SolrException",
      "root-error-class","org.apache.solr.common.SolrException"},
      "msg":"The field location does not support spatial filtering",
      "trace":["org.apache.solr.common.SolrException: The field location does not support spatial filtering\r\n\tat
org.apache.solr.search.SpatialFilterParser.parse(SpatialFilterParser.java:82)\r\n\tat org.apache.solr.search.QParser.getQuery(QParser.java:148)\r\n\tat
org.apache.solr.handler.component.QueryComponent.prepare(QueryComponent.java:212)\r\n\tat org.apache.solr.handler.component.SearchHandler.handleRequestBody(SearchHandler.java:269)\r\n\tat
org.apache.solr.handler.RequestHandlerBase.handleRequest(RequestHandlerBase.java:173)\r\n\tat org.apache.solr.core.SolrCore.execute(SolrCore.java:2440)\r\n\tat
org.apache.solr.servlet.HttpSolrCall.execute(HttpSolrCall.java:723)\r\n\tat org.apache.solr.servlet.HttpSolrCall.call(HttpSolrCall.java:529)\r\n\tat
org.apache.solr.servlet.SolrDispatchFilter.doFilter(SolrDispatchFilter.java:247)\r\n\tat org.apache.solr.servlet.SolrDispatchFilter.doFilter(SolrDispatchFilter.java:298)\r\n\tat
org.eclipse.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1491)\r\n\tat org.eclipse.jetty.servlet.ServletHandler.doHandle(ServletHandler.java:502)\r\n\tat
org.eclipse.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:143)\r\n\tat org.eclipse.jetty.security.SecurityHandler.handle(SecurityHandler.java:548)\r\n\tat
org.eclipse.jetty.server.session.SessionHandler.doHandle(SessionHandler.java:226)\r\n\tat org.eclipse.jetty.server.handler.ContextHandler.doHandle(ContextHandler.java:1155)\r\n\tat
org.eclipse.jetty.servlet.ServletHandler.doScope(ServletHandler.java:321)\r\n\tat org.eclipse.jetty.server.session.SessionHandler.doScope(SessionHandler.java:151)\r\n\tat
org.eclipse.jetty.server.handler.ContextHandler.doScope(ContextHandler.java:1122)\r\n\tat org.eclipse.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:143)\r\n\tat
org.eclipse.jetty.server.handler.HandlerCollection.handle(HandlerCollection.java:213)\r\n\tat org.eclipse.jetty.server.handler.HandlerCollection.handle(HandlerCollection.java:119)\r\n\tat
org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:134)\r\n\tat org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:134)\r\n\tat
org.eclipse.jetty.server.Server.handle(Server.java:551)\r\n\tat org.eclipse.jetty.server.HttpChannel.handle(HttpChannel.java:320)\r\n\tat
org.eclipse.jetty.server.HttpConnection.onFillable(HttpConnection.java:251)\r\n\tat org.eclipse.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:273)\r\n\tat
org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:95)\r\n\tat org.eclipse.jetty.io.SelectChannelEndPoint$2.run(SelectChannelEndPoint.java:93)\r\n\tat
org.eclipse.jetty.util.thread.strategy.ExecuteProduceConsume.executeProduceConsume(ExecuteProduceConsume.java:303)\r\n\tat
org.eclipse.jetty.util.thread.strategy.ExecuteProduceConsume.produceConsume(ExecuteProduceConsume.java:148)\r\n\tat
org.eclipse.jetty.util.thread.strategy.ExecuteProduceConsume.run(ExecuteProduceConsume.java:136)\r\n\tat org.eclipse.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:712)\r\n\tat
org.eclipse.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:589)\r\n\tat java.lang.Thread.run(Thread.java:745)\r\n",
      "code":500}}

```

El problema que se presenta se cree que es porque en schema a pesar de crear como corresponde el tipo de campo “location” y el campo dinámico “_coordinate” es que en el libro no se menciona la creación del campo mismo que llame al tipo campo “location”.

Inspeccionando el schema veo que luego de indexar se actualiza y aparece un campo llamado “location”, sin embargo, es de tipo string, siendo que debiese ser de tipo “location”. Lo que tuve que hacer fue antes de la indexación, crear el campo de nombre “location” de tipo “location”

```
<field name="location" type="location" indexed="true" stored="true"/>
```

Un indicio de que las cosas van mejorando es esto:

Request-Handler (qt)

/select

common

q

,

fq

location pt=37.775,-122.419 d=20}

sort

start, rows

0 10

fl

df

http://localhost:8983/solr/search_gspatial/select?fq={!geofilt sfield=location

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "q": "*,*",
      "indent": "on",
      "fq": "{!geofilt sfield=location pt=37.775,-122.419 d=20}",
      "wt": "json",
      "_: "1495774151717"}},
  "response": { "numFound": 1, "start": 0, "docs": [
    {
      "id": "3",
      "location": "37.775,-122.419",
      "city": ["San Francisco, CA"],
      "location_0_coordinate": 37.775,
      "location_1_coordinate": -122.419,
      "_version_": 1568432790269067264}]
  }
}
```

Como podemos ver, al parecer si se está trabajando con el campo de forma correcta dado que al recibir el par de coordenadas se acudió a la clase LatLonType ya que ella es quien se encarga de separar el par de coordenadas considerándolas como campos distintos.

Aun así no se muestran los resultados que se esperan ya que no arroja ningún documento la búsqueda, siendo que como mínimo debiese de mostrar un documento que es el que estamos buscando.

Ahora la duda es como realizar esta consulta de forma correcta:

http://localhost:8983/solr/geospatial/select?q=*&fq={!geofilt sfield=location pt=37.775,-122.419 d=20}

Request-Handler (qt)

/select

common

q

fq={!geofilt sfield=location pt=37.775,-122.419 d=20}

fq

sort

start, rows

0 10

fl

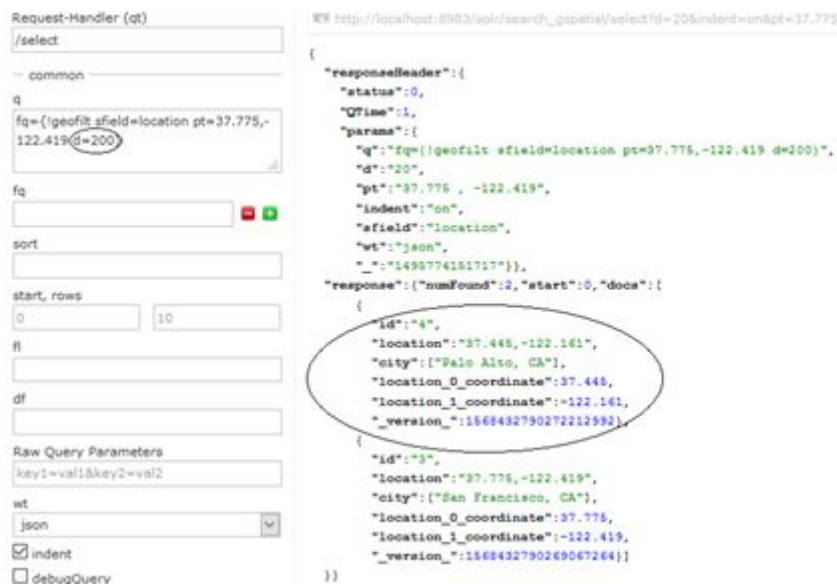
df

Raw Query Parameters

http://localhost:8983/solr/search_gspatial/select?d=20&indent=on&pt=37.775

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 3,
    "params": {
      "q": "fq={!geofilt sfield=location pt=37.775,-122.419 d=20}",
      "d": "20",
      "pt": "37.775 , -122.419",
      "indent": "on",
      "sfield": "location",
      "wt": "json",
      "_: "1495774151717"}},
  "response": { "numFound": 1, "start": 0, "docs": [
    {
      "id": "3",
      "location": "37.775,-122.419",
      "city": ["San Francisco, CA"],
      "location_0_coordinate": 37.775,
      "location_1_coordinate": -122.419,
      "_version_": 1568432790269067264}]
  }
}
```

Sin embargo, encuentro que el ejemplo no llama mucho la atención dado que la distancia de 20 km que se ingresó en la consulta es demasiado pequeña y filtró todos los documentos menos al que nos referíamos por lo que decidí aumentar la distancia.



Ahora si!! Podemos apreciar que al dar una distancia de 200 km nos encontramos con un documento que si está en el radio, se trata de "Palo Alto, CA"

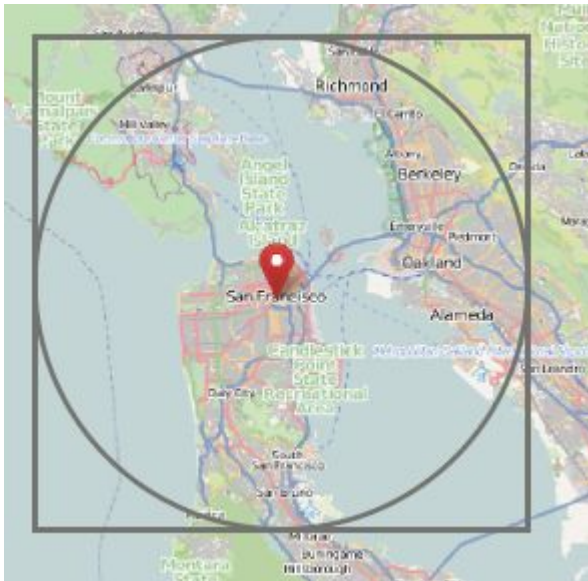
Definiendo los atributos utilizados:

- sfield: se debe mencionar cuál es el campo espacial compatible con LatOnType
- pt: son las coordenadas que vienen de forma latitud,longitud
- d: es la distancia en km que fija el tamaño del radio de cercanía
- geofilt: filtro de búsqueda espacial para encontrar documentos cercanos, basado en radio.

Funcionamiento de geofilt:

Se identifican dos pasos, primero crea un cuadro delimitador determinado por la distancia que se ingresa a la consulta para considerar solo los que estén dentro del área. El segundo paso es establecer un delimitador circular al interior del cuadro delimitador con radio igual a la distancia ingresada. Los documentos que estén dentro del cuadro delimitador pero fuera del círculo serán filtrados. Para evaluar si están dentro de este círculo se deben calcular las distancias de todos los documentos que están dentro del cuadrado.

Se dice que este filtro se ocupa para cuando tenemos muchos documentos y necesitamos mayor precisión pero si no es el caso, no es necesario ocuparlo dado que pasa a ser costoso en comparación a la diferencia con el cuadro delimitador que se realizó en el primer paso y la diferencia de precisión podría expresarse en metros.



Funcionamiento de bbox:

El primer paso realizado en geofilt es lo que realiza bbox y tiene la misma forma de consulta.

/select

common

q

`fq={bbox sfield=location pt=37.775,-122.419 d=200}`

fq

sort

start, rows

0 10

fi

df

Raw Query Parameters

key1=val1&key2=val2

wt

json

☒ indent

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 147,
    "params": {
      "q": "fq={bbox sfield=location pt=37.775,-122.419 d=200}",
      "d": "20",
      "pt": "37.775 , -122.419",
      "indent": "on",
      "sfield": "location",
      "wt": "json",
      "_": "1495774151717"
    }
  },
  "response": { "numFound": 2, "start": 0, "docs": [
    {
      "id": "3",
      "location": "37.775,-122.419",
      "city": ["San Francisco, CA"],
      "location_0_coordinate": 37.775,
      "location_1_coordinate": -122.419,
      "_version_": 1568432790269067264
    },
    {
      "id": "4",
      "location": "37.445,-122.161",
      "city": ["Palo Alto, CA"],
      "location_0_coordinate": 37.445,
      "location_1_coordinate": -122.161,
      "_version_": 1568432790272212992
    }
  ]
}
```

Aprovechando el cálculo de las distancias de los documentos (en gfilt)

Retorno de las distancias calculadas en los resultados de búsqueda

Geodist es una función que devuelve la distancia entre el punto que buscamos y demás documentos.

La sintaxis de la función geodist es geodist (sfield, latitud, longitud). Esta función devuelve un pseudo campo con el valor de la distancia.

No me reconoce el campo distance, presumiblemente por que se genera con el campo dinamico resultante de la función, por lo que no lo pude ver en panel de consultas pero si por el navegador.

[http://localhost:8983/solr/geospatial/select?q=*&fl=id,city,distance:geodist\(location,37.77493,-122.41942\)](http://localhost:8983/solr/geospatial/select?q=*&fl=id,city,distance:geodist(location,37.77493,-122.41942))



```

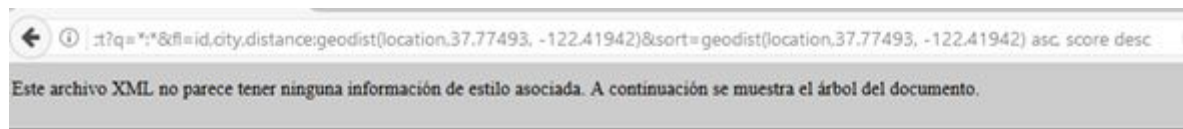
- <response>
  - <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">0</int>
    - <lst name="params">
      <str name="q">*</str>
      - <str name="fl">
        id,city,distance:geodist(location,37.77493,-122.41942)
      </str>
    </lst>
  </lst>
  - <result name="response" numFound="4" start="0">
    - <doc>
      <str name="id">1</str>
      - <arr name="city">
        <str>Atlanta, GA</str>
      </arr>
      <double name="distance">3436.669993915123</double>
    </doc>
    - <doc>
      <str name="id">2</str>
      - <arr name="city">
        <str>New York, NY</str>
      </arr>
      <double name="distance">4128.9603389283575</double>
    </doc>
  </result>
</response>

```

Ordenamiento de los resultados por distancias calculadas

[http://localhost:8983/solr/search_gspatial/select?q=*&fl=id,city,distance:geodist\(location,37.77493,-122.41942\)&sort=geodist\(location,37.77493,-122.41942\) asc, score desc](http://localhost:8983/solr/search_gspatial/select?q=*&fl=id,city,distance:geodist(location,37.77493,-122.41942)&sort=geodist(location,37.77493,-122.41942) asc, score desc)

Esta solicitud clasificará todos los documentos desde la menor distancia hasta el más grande, y luego por puntuación de mayor a menor (si hay una puntuación de relevancia significativa, a diferencia de este ejemplo).



```

- <response>
+ <lst name="responseHeader"></lst>
- <result name="response" numFound="4" start="0">
  - <doc>
    <str name="id">3</str>
    - <arr name="city">
      <str>San Francisco, CA</str>
    </arr>
    <double name="distance">0.03772596784117343</double>
  </doc>
  - <doc>
    <str name="id">4</str>
    - <arr name="city">
      <str>Palo Alto, CA</str>
    </arr>
    <double name="distance">43.17493506307893</double>
  </doc>
  - <doc>
    <str name="id">1</str>
    - <arr name="city">
      <str>Atlanta, GA</str>
    </arr>
    <double name="distance">3436.669993915123</double>
  </doc>

```

Búsqueda espacial avanzada:

Hasta ahora hemos visto como realizar una búsqueda espacial basándonos en un solo punto indexando documentos que contienen solo un campo de ubicación.

Existen búsquedas más avanzadas que permiten indexar documentos con campos que contienen polígonos arbitrarios que representan su ubicación (en vez de un solo punto), permitiendo la indexación de múltiples puntos o formas por campo.

Se menciona un ejemplo sobre una cadena de restaurantes para captar la utilidad que este tipo de búsquedas puede tener, la cadena de restaurantes estará ubicada en distintas localidades y tendríamos que crear distintos documentos que contengan su ubicación en vez de crear solo un documento que contenga.

La funcionalidad de poder representar ubicaciones a través de círculos o polígonos tiene la funcionalidad de adecuarse a las características geográficas de lo que se requiere representar.

Esto lo permite la clase `SpatialRecursivePrefixTreeFieldType`

Formas que pueden representar ubicaciones en este tipo de búsquedas:

Un punto:

```
<field name="location_rpt">43.17614,-90.57341</field>
```

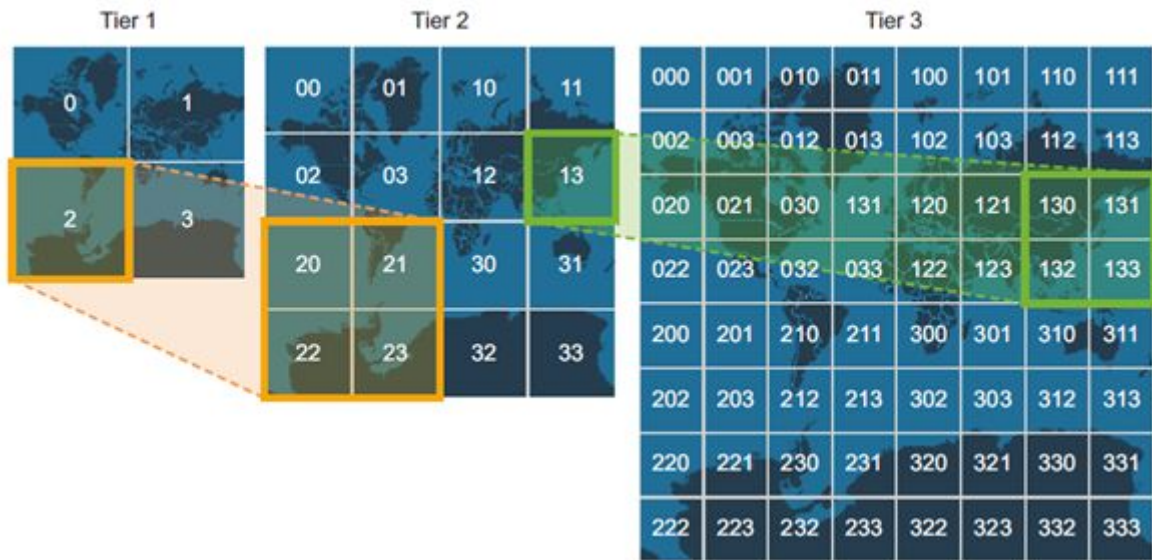
Un polígono:

```
<field name="location_rpt">-74.093 41.042 -69.347 44.558</field>
```

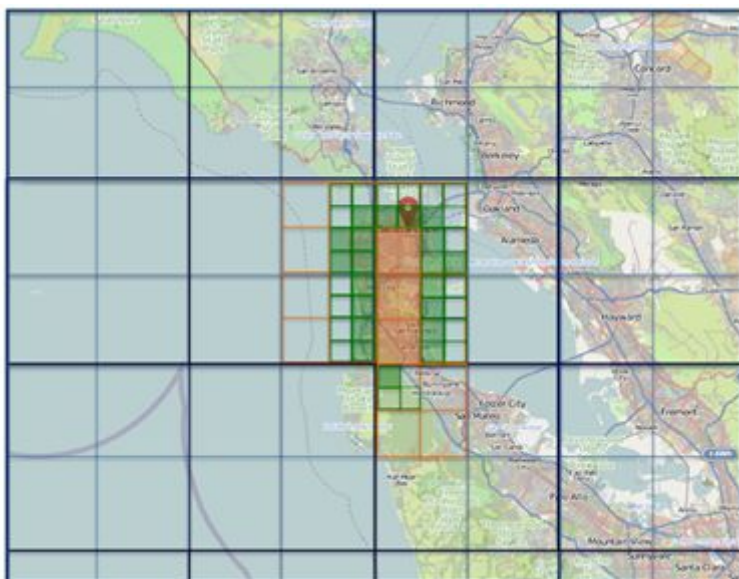
Un Círculo:

```
<field name="location_rpt">Circle(37.775,-122.419 d=20)</field>
```

Al decir que un campo puede contener polígonos en su interior estamos asumiendo que podemos representar un área de forma de sub áreas.



Lo interesante de esto es que las formas pueden adecuarse a la geografía de lo que queremos representar.



3. Elasticsearch

3.1. Explorando el Cluster

Para iniciar Elasticsearch se debe ir a la carpeta bin (en la terminal) y ejecutar el archivo elasticsearch:

```
./elasticsearch
```

Elasticsearch cuenta con una REST API para poder hacer las peticiones. De esta forma se puede ver el estado del cluster con el siguiente comando:

```
curl -XGET 'localhost:9200/_cat/health?v&pretty'
```

```
epoch      timestamp cluster      status node.total node.data shards pri relo init u
nassign pending_tasks max_task_wait_time active_shards_percent
1497784000 07:06:40  elasticsearch yellow          1          1        6  6    0    0
              6              0              -          50.0%
```

Para poder ver los nodos del cluster se utiliza el siguiente comando (notar que en la imagen anterior se indicaba que existía un nodo):

```
curl -XGET 'localhost:9200/_cat/nodes?v&pretty'
```

```
ip          heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
127.0.0.1   10           96    5   0.10    0.34    0.53 mdi      *     W10F
PZ7
```

Para poder ver los índices:

```
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
```

```
health status index      uuid                pri rep docs.count docs.deleted store.size pri.store.size
yellow open   .kibana  0vcx2bKyQxOY0uyUsihns  1  1      1          0      3.2kb      3.2kb
yellow open   customer xg0uoNJ2SlCRAWPPyM4w_Q  5  1      1          0       4kb       4kb
```

Al ejecutar el comando se pueden ver los dos índices que existen, “customer”, creado previamente y .kibana el cual fue creado al ejecutar Kibana.

3.2. Indexación y Búsqueda

Para crear un índice se debe utilizar el siguiente comando:

```
curl -XPUT 'localhost:9200/cliente?pretty&pretty'
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
```

El primer comando crea el índice mientras (como se vio anteriormente) el segundo lista los índices.

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true
}
```

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	.kibana	0vcx2bKyQx0Y0uyUsihnsG	1	1	1	0	3.2kb	3.2kb
yellow	open	cliente	PMJ_795jQk6HW8ZAQF9JYA	5	1	0	0	650b	650b
yellow	open	customer	xg0uoNJ2SLCRAWPPyM4w_Q	5	1	1	0	4kb	4kb

Para indexar un documento se utiliza el siguiente comando:

```
curl -XPUT 'localhost:9200/cliente/external/1?pretty&pretty' -H 'Content-Type: application/json' -d'
```

```
{
  "name": "Juan Nieves"
}
```

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "created" : true
}
```

Para buscar el documento recién indexado:

```
curl -XGET 'localhost:9200/cliente/external/1?pretty&pretty'
```

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : {
    "name" : "Juan Nieves"
  }
}
```

Para borrar un índice:

```
curl -XDELETE 'localhost:9200/customer?pretty&pretty'
```

```
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
```

```
{
  "acknowledged" : true
}
```

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	.kibana	0vcx2bKyQx0Y0uyUsihnsG	1	1	1	0	3.2kb	3.2kb
yellow	open	cliente	PMJ_795jQk6Hw8ZAQF9JYA	5	1	1	0	3.9kb	3.9kb

Para modificar los datos de un documento se utiliza el verbo PUT, de la misma forma que indexamos un documento, indicando el ID del documento previamente indexado:

```
curl -XPUT 'localhost:9200/cliente/external/1?pretty&pretty' -H 'Content-Type: application/json' -d'
```

```
{
  "name": "Juana Nieves"
}
```

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "1",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "created" : false
}
```

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "1",
  "_version" : 2,
  "found" : true,
  "source" : {
    "name" : "Juana Nieves"
  }
}
```

Notar que al usar el verbo PUT, se debe especificar el id del documento, mientras que al utilizar el verbo POST, no es necesario (se crea un nuevo documento):

```
curl -XPOST 'localhost:9200/cliente/external?pretty&pretty' -H 'Content-Type: application/json' -d'
```

```
{
  "name": "Juan Nieves"
}
```

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "AVy7cm9TK6njVBQ1uxUK",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "created" : true
}
```


Para actualizar documentos, Elasticsearch realmente borra el anterior y vuelve a ingresar uno nuevo. Como ejemplo se agregara un nuevo campo al documento con el cliente “Juana Nieves”:

```
curl -XPOST 'localhost:9200/cliente/external/1/_update?pretty&pretty' -H 'Content-Type: application/json' -d'
```

```
{
  "doc": { "name": "Juana Nieves", "edad": 20 }
},
```

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "1",
  "_version" : 3,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  }
}
```

o bien utilizando un script:

```
curl -XPOST 'localhost:9200/cliente/external/1/_update?pretty&pretty' -H 'Content-Type: application/json' -d'
```

```
{
  "script" : "ctx._source.edad += 5"
},
```

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "1",
  "_version" : 4,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  }
}
```

El script realizado modifica la edad del cliente sumando cinco años, esto es posible ya que ctx._source hace referencia al documento que se especifica (documento con ID 1).

```
{
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "1",
  "_version" : 4,
  "found" : true,
  "_source" : {
    "name" : "Juana Nieves",
    "edad" : 25
  }
}
```

Para borrar un documento:

`curl -XDELETE 'localhost:9200/cliente/external/2?pretty&pretty'`

```
{
  "found" : true,
  "_index" : "cliente",
  "_type" : "external",
  "_id" : "2",
  "_version" : 2,
  "result" : "deleted",
  "shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  }
}
```

Elasticsearch permite realizar operaciones múltiples en una sola acción, el siguiente comando indexa dos documentos en una sola acción (_bulk):

```
curl -XPOST 'localhost:9200/cliente/external/_bulk?pretty&pretty' -H 'Content-Type: application/json' -d'
{"index":{"_id":"3"}}
{"name": "Catherine Rojas" }
{"index":{"_id":"4"}}
{"name": "Alfonso Tapia" }
'
```

```
{
  "took" : 170,
  "errors" : false,
  "items" : [
    {
      "index" : {
        "_index" : "cliente",
        "_type" : "external",
        "_id" : "3",
        "_version" : 1,
        "result" : "created",
        "shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        },
        "created" : true,
        "status" : 201
      }
    },
    {
      "index" : {
        "_index" : "cliente",
        "_type" : "external",
        "_id" : "4",
        "_version" : 1,
        "result" : "created",
        "shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        },
        "created" : true,
        "status" : 201
      }
    }
  ]
}
```

3.3. Explorando el Contenido

En este ejemplo indexaremos una mayor cantidad de datos. El archivo accounts.json posee documentos con la siguiente estructura:

```
{
  "account_number": 0,
  "balance": 16623,
  "firstname": "Bradshaw",
  "lastname": "Mckenzie",
  "age": 29,
  "gender": "F",
  "address": "244 Columbus Place",
  "employer": "Euron",
  "email": "bradshawmckenzie@euron.com",
  "city": "Hobucken",
  "state": "CO"
}
```

Para indexar el documento json:

```
curl -H "Content-Type: application/json" -XPOST 'localhost:9200/bank/account/_bulk?pretty&refresh'
--data-binary "@accounts.json"
```

Donde los documentos se indexaran en el indice “bank”, bajo el tipo “account”:

```
{
  "created" : true,
  "status" : 201
},
{
  "index" : {
    "index" : "bank",
    "type" : "account",
    "id" : "995",
    "version" : 1,
    "result" : "created",
    "forced_refresh" : true,
    "shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
    },
    "created" : true,
    "status" : 201
  }
}
```

Para ver el indice:

```
curl 'localhost:9200/_cat/indices?v'
```

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	.kibana	0vcx2bKyQxOY0uyUsthnsg	1	1	1	0	3.2kb	3.2kb
yellow	open	bank	2AyqsvF0QZGC1qXozcXJfw	5	1	1000	0	640.3kb	640.3kb
yellow	open	cliente	PMJ_795jQk6HW8ZAQF9JYA	5	1	4	0	14.3kb	14.3kb

El resultado muestra que el indice bank contiene 1000 documentos indexados.

Para realizar las búsquedas existen dos metodos “REST request URI” y “REST request body”. Un ejemplo del primer metodo es el comando siguiente:

```
curl -XGET 'localhost:9200/bank/_search?q=*&sort=account_number:asc&pretty&pretty'
```

El cual realiza una búsqueda general, ordenando el resultado ascendentemente por el numero de cuenta:

```
{
  "took" : 280,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : null,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "0",
        "_score" : null,
        "_source" : {
          "account_number" : 0,
          "balance" : 16623,
          "firstname" : "Bradshaw",
          "lastname" : "Mckenzie",
          "age" : 29,
          "gender" : "F",
          "address" : "244 Columbus Place",
          "employer" : "Euron",
          "email" : "bradshawmckenzie@euron.com",
          "city" : "Hobucken",
          "state" : "CO"
        },
        "sort" : [
          0
        ]
      }
    ]
  }
}
```

El segundo método es el utilizado en los casos anteriores:

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" }
  ]
}
```

Utilizando los filtros de las búsquedas podemos realizar consultas como las siguientes:

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": { "match_all": {} },
  "from": 10,
  "size": 10
}
```

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}
```



```
{
  "took" : 7,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 7.3900023,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "136",
        "_score" : 7.3900023,
        "_source" : {
          "account_number" : 136,
          "balance" : 45801,
          "firstname" : "Winnie",
          "lastname" : "Holland",
          "age" : 38,
          "gender" : "M",
          "address" : "198 Mill Lane",
          "employer" : "Neteria",
          "email" : "winnieholland@neteria.com",
          "city" : "Urie",
          "state" : "IL"
        }
      }
    ]
  }
}
```

3.4. Relevancia (Score)

Al realizar una búsqueda en Elasticsearch, este en el resultado obtenido entrega un campo llamado "score", el cual indica la relevancia del documento respecto a la búsqueda realizada. Así mientras más grande es el score del documento más relevancia tiene. Pero no siempre se necesario determinar el score. Elasticsearch se da cuenta de estas situaciones y solo realiza el score cuando es necesario.

Se usará el archivo shakespeare.json


```
curl -XPUT 'localhost:9200/shakespeare?pretty' -H 'Content-Type: application/json' -d'
{
  "mappings" : {
    "_default_" : {
      "properties" : {
        "speaker" : { "type": "keyword" },
        "play_name" : { "type": "keyword" },
        "line_id" : { "type": "integer" },
        "speech_number" : { "type": "integer" }
      }
    }
  }
}
```

Se realiza una búsqueda:

```
curl -XGET 'localhost:9200/shakespeare/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": { "match": { "text_entry": "blessed" } }
}
```

```
{
  "took" : 590,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 70,
    "max_score" : 10.188129,
    "hits" : [
      {
        "_index" : "shakespeare",
        "_type" : "line",
        "_id" : "86280",
        "_score" : 10.188129,
        "_source" : {
          "line_id" : 86281,
          "play_name" : "Romeo and Juliet",
          "speech_number" : 33,
          "line_number" : "2.2.145",
          "speaker" : "ROMEO",
          "text_entry" : "O blessed, blessed night! I am afeard."
        }
      },
      {
        "_index" : "shakespeare",
        "_type" : "line",
        "_id" : "42998",
        "_score" : 9.024506,
        "_source" : {
          "line_id" : 42999,
          "play_name" : "Henry VIII",
          "speech_number" : 44,
          "line_number" : "5.1.190",
          "speaker" : "Old Lady",
          "text_entry" : "Under their blessed wings!"
        }
      },
      {
        "_index" : "shakespeare",
```

Otras búsquedas:

```
curl -XGET 'localhost:9200/shakespeare/_search?pretty' -H 'Content-Type: application/json'
-d'
{
  "query": { "match_phrase": { "text_entry": "how deep" } }
}
```

3.5. Teoría detrás de la Relevancia (Scoring)

Lucene (y por lo tanto Elasticsearch) utiliza el modelo booleano para encontrar documentos coincidentes, y una fórmula llamada función de puntuación práctica para calcular la relevancia. Esta fórmula toma prestados conceptos a partir de frecuencia de término / frecuencia de documento inverso y el modelo de espacio vectorial, pero añade características más modernas como un factor de coordinación, normalización de longitud de campo y aumento de cláusulas de términos o consultas.

Una vez que tengamos una lista de documentos coincidentes, deben ser clasificados por relevancia. No todos los documentos contienen todos los términos, y algunos términos son más importantes que otros. La puntuación de relevancia de todo el documento depende (en parte) del peso de cada término de consulta que aparece en ese documento.

El peso de un término está determinado por tres factores:

Frecuencia de término

¿Con qué frecuencia aparece el término en el campo? Cuanto más a menudo, más relevante. Un campo que contiene cinco menciones del mismo término es más probable que sea relevante que un campo que contiene sólo una mención. El término frecuencia se calcula de la siguiente manera:

$$tf(t \text{ in } d) = \sqrt{\text{frequency}}$$

El término frecuencia (tf) para el término t en el documento d es la raíz cuadrada del número de veces que el término aparece en el documento.

Frecuencia inversa del documento

¿Con qué frecuencia aparece cada término en el índice? Cuanto más a menudo, menos relevante. Los términos que aparecen en muchos documentos tienen un peso menor que los términos más raros. La frecuencia del documento inverso se calcula de la siguiente manera:

$$idf(t) = 1 + \log (\text{numDocs} / (\text{docFreq} + 1))$$

La frecuencia del documento inverso (idf) del término t es el logaritmo del número de documentos en el índice, dividido por el número de documentos que contienen el término.

Norma campo-longitud

¿Que tan largo es el campo? Cuanto más largo es, menos probable es que las palabras en el campo sean relevantes. Un término que aparece en un campo de título corto lleva más peso que el mismo término que aparece en un campo de contenido largo. La norma de longitud de campo se calcula de la siguiente manera:

$$\text{norm}(d) = 1 / \sqrt{\text{numTerms}}$$

La norma campo-longitud (norma) es la raíz cuadrada inversa del número de términos en el campo.

Estos tres factores -la frecuencia de los términos, la frecuencia del documento inverso y la norma de la longitud del campo- se calculan y almacenan a tiempo de indexación. En conjunto, se utilizan para calcular el peso de un solo término en un documento en particular.

Ejemplo búsqueda “blessed”

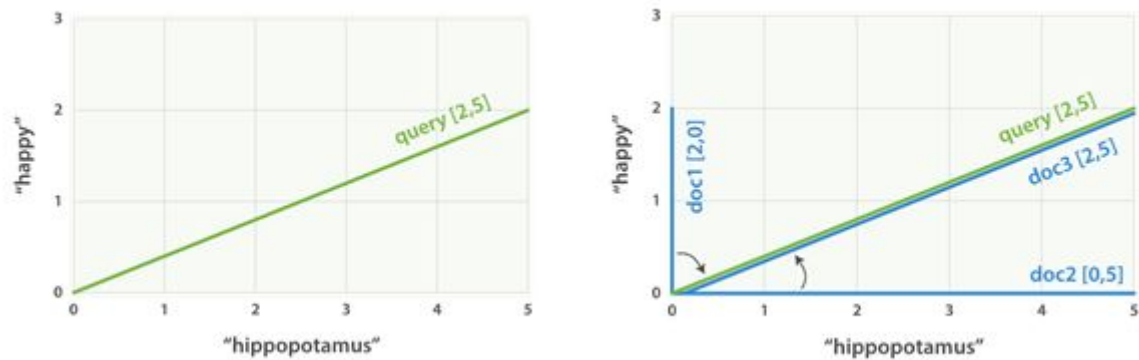
```
{
  "shard" : "[shakespeare][1]",
  "node" : "W10FPZ7QRFOHlwTIwn86w",
  "index" : "shakespeare",
  "type" : "line",
  "id" : "86280",
  "score" : 10.188129,
  "source" : {
    "line_id" : 86281,
    "play_name" : "Romeo and Juliet",
    "speech_number" : 33,
    "line_number" : "2.2.145",
    "speaker" : "ROMEO",
    "text_entry" : "O blessed, blessed night! I am afeard."
  },
  "explanation" : {
    "value" : 10.18813,
    "description" : "weight(text_entry:blessed in 11909) [PerFieldSimilarity], result of:",
    "details" : [
      {
        "value" : 10.18813,
        "description" : "score(doc=11909, freq=2.0 = termFreq=2.0\n), product of:",
        "details" : [
          {
            "value" : 7.34512,
            "description" : "idf, computed as log(1 + (docCount - docFreq + 0.5) / (docFreq + 0.5)) from:",
            "details" : [
              {
                "value" : 14.0,
                "description" : "docFreq",
                "details" : [ ]
              },
              {
                "value" : 22454.0,
                "description" : "docCount",
                "details" : [ ]
              }
            ]
          }
        ]
      },
      {
        "value" : 1.3870611,
        "description" : "tfNorm, computed as (freq * (k1 + 1)) / (freq + k1 * (1 - b + b * fieldLength / avgFieldLength)) from:",

```

Modelo de espacio vectorial

El modelo de espacio vectorial proporciona una forma de comparar una consulta multi término con un documento. La salida es una puntuación única que representa lo bien que el documento coincide con la consulta. Para ello, el modelo representa tanto el documento y la consulta como vectores.

Un vector es realmente sólo una matriz unidimensional que contiene números. En el modelo de espacio vectorial, cada número en el vector es el peso de un término.



Al realizar la búsqueda “how deep”

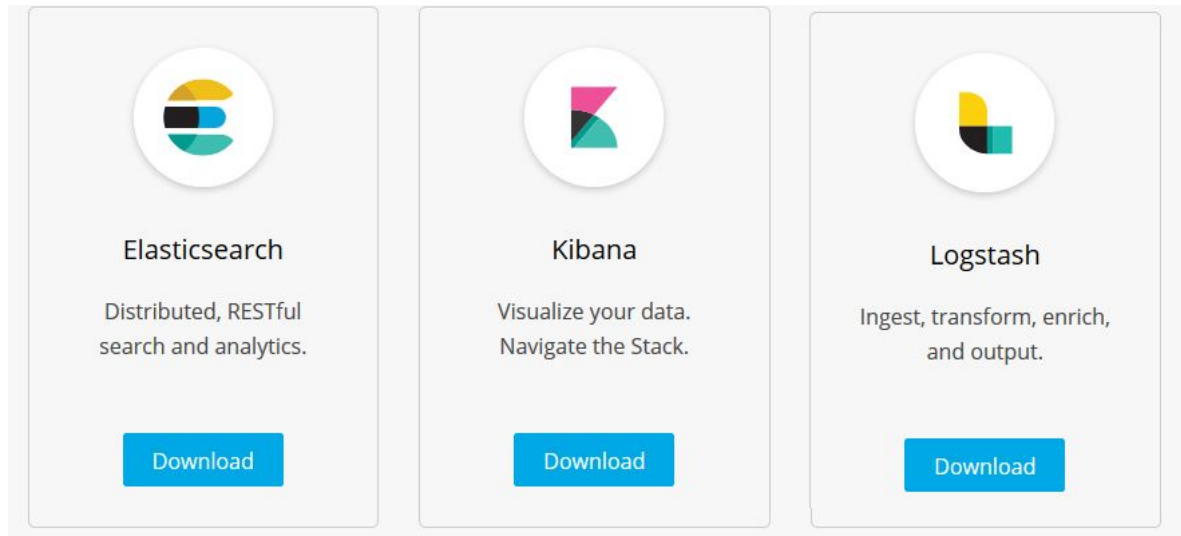
```

"value" : 14.600049,
"description" : "score(doc=2822,freq=1.0 = phraseFreq=1.0\n), product of:",
"details" : [
  {
    "value" : 10.711014,
    "description" : "idf(), sum of:",
    "details" : [
      {
        "value" : 4.0416493,
        "description" : "idf, computed as log(1 + (docCount - docFreq + 0.5) / (docFreq + 0.5)) from:",
        "details" : [
          {
            "value" : 394.0,
            "description" : "docFreq",
            "details" : [ ]
          },
          {
            "value" : 22454.0,
            "description" : "docCount",
            "details" : [ ]
          }
        ]
      },
      {
        "value" : 0.6693645,
        "description" : "idf, computed as log(1 + (docCount - docFreq + 0.5) / (docFreq + 0.5)) from:",
        "details" : [
          {
            "value" : 28.0,
            "description" : "docFreq",
            "details" : [ ]
          },
          {
            "value" : 22454.0,
            "description" : "docCount",
            "details" : [ ]
          }
        ]
      }
    ]
  },
  {
    "value" : 1.3630875,

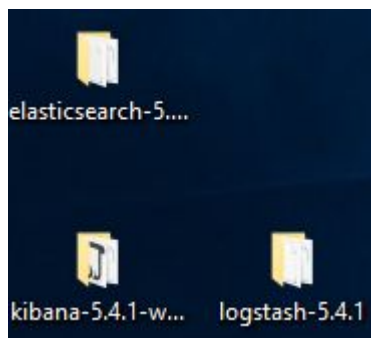
```

4. Logstash y Kibana

Descargar las tres herramientas a utilizar desde <https://www.elastic.co/downloads>



Descomprimir archivos donde se desee, en este caso el escritorio.



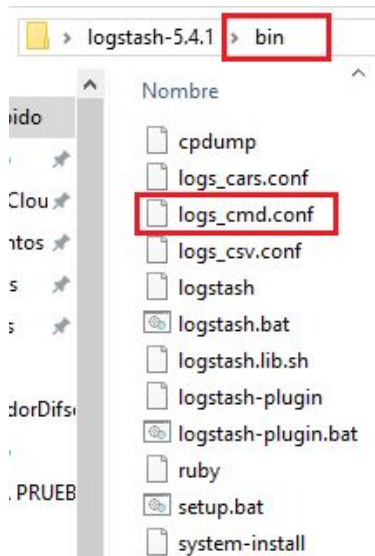
Esta parte se divide en 3 partes:

1. Indexar y visualizar logs por cmd a través de logstash
2. Indexar y visualizar logs obtenidos desde un archivo .csv a través de logstash
3. Indexar y visualizar logs obtenidos desde un .csv generado desde apache jmeter al realizar una petición http a un sitio web.

4.1. Indexar y visualizar logs por cmd

Para comenzar lo que haremos es crear logs desde consola, enviarlos a elasticsearch y visualizarlos en kibana.

Para crear logs por consola debemos ingresar a la carpeta de logstash\bin y crear un archivo de configuración:



```
logs_cmd.conf
input{
  stdin {}
}

filter{
}

output{
  elasticsearch{
    hosts => ["localhost:9200"]
    index => "primer_indice"
  }
  stdout{}
}
```

En input-> stdin indica que los datos serán obtenidos por teclado.

En output -> se indica que se hará conexión con elasticsearch, se debe indicar el host que utilizamos para trabajar con elasticsearch y debemos especificar el nombre que tendrá el índice

Ejecutamos elasticsearch con el comando:

```
C:\Users\Crsthian\Desktop\elasticsearch-5.4.1\bin>elasticsearch.bat
[2017-07-19T13:49:00,761][INFO ][o.e.n.Node               ] [] initializing ...
[2017-07-19T13:49:01,632][INFO ][o.e.e.NodeEnvironment ] [m_05U29] using [1] data paths, mounts [[OS (C:)], net usab
le space [235.9gb], net total_space [372.6gb], spins? [unknown], types [NTFS]]
[2017-07-19T13:49:01,633][INFO ][o.e.e.NodeEnvironment ] [m_05U29] heap size [1.9gb], compressed ordinary object poin
ters [true]
[2017-07-19T13:49:01,651][INFO ][o.e.n.Node               ] node name [m_05U29] derived from node ID [m_05U29bQH-Chal0z9
AkNw]; set [node.name] to override
[2017-07-19T13:49:01,653][INFO ][o.e.n.Node               ] version[5.4.1], pid[15096], build[2cfe0df/2017-05-29T16:05:5
1.443Z], OS[Windows 10/10.0/amd64], JVM[Oracle Corporation/Java HotSpot(TM) 64-Bit Server VM/1.8.0_121/25.121-b13]
[2017-07-19T13:49:01,655][INFO ][o.e.n.Node               ] JVM arguments [-Xms2g, -Xmx2g, -XX:+UseConcMarkSweepGC, -XX:
CMSInitiatingOccupancyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -XX:+DisableExplicitGC, -XX:+AlwaysPreTouch, -Xss
1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djna.nosys=true, -Djdk.io.permissionsUseCanonicalPath=true, -Dio.n
etty.noUnsafe=true, -Dio.netty.noKeySetOptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0, -Dlog4j.shutdownHo
okEnabled=false, -Dlog4j2.disable.jmx=true, -Dlog4j.skipJansi=true, -XX:+HeapDumpOnOutOfMemoryError, -Delasticsearch, -D
es.path.home=C:\Users\Crsthian\Desktop\elasticsearch-5.4.1]
[2017-07-19T13:49:05,526][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [aggs-matrix-stats]
[2017-07-19T13:49:05,531][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [ingest-common]
[2017-07-19T13:49:05,532][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [lang-expression]
[2017-07-19T13:49:05,532][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [lang-groovy]
[2017-07-19T13:49:05,532][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [lang-mustache]
[2017-07-19T13:49:05,533][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [lang-painless]
[2017-07-19T13:49:05,535][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [percolator]
[2017-07-19T13:49:05,536][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [reindex]
[2017-07-19T13:49:05,537][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [transport-netty3]
[2017-07-19T13:49:05,539][INFO ][o.e.p.PluginsService     ] [m_05U29] loaded module [transport-netty4]
[2017-07-19T13:49:05,542][INFO ][o.e.p.PluginsService     ] [m_05U29] no plugins loaded
[2017-07-19T13:49:14,562][INFO ][o.e.d.DiscoveryModule ] [m_05U29] using discovery type [zen]
[2017-07-19T13:49:15,775][INFO ][o.e.n.Node               ] initialized
[2017-07-19T13:49:15,776][INFO ][o.e.n.Node               ] [m_05U29] starting ...
[2017-07-19T13:49:17,197][INFO ][o.e.t.TransportService ] [m_05U29] publish_address {127.0.0.1:9300}, bound_addresses
{127.0.0.1:9300}, {[::1]:9300}
[2017-07-19T13:49:20,411][INFO ][o.e.c.s.ClusterService ] [m_05U29] new_master {m_05U29bQH-Chal0z9AkNw}{H9d2C
GLOQhmwzKC2uTfZvg}{127.0.0.1}{127.0.0.1:9300}, reason: zen-disco-elected-as-master ([0] nodes joined)
[2017-07-19T13:49:20,587][INFO ][o.e.g.GatewayService   ] [m_05U29] recovered [0] indices into cluster state
[2017-07-19T13:49:21,678][INFO ][o.e.h.n.Netty4HttpServerTransport] [m_05U29] publish_address {127.0.0.1:9200}, bound_ad
dresses {127.0.0.1:9200}, {[::1]:9200}
[2017-07-19T13:49:21,701][INFO ][o.e.n.Node               ] [m_05U29] started
```

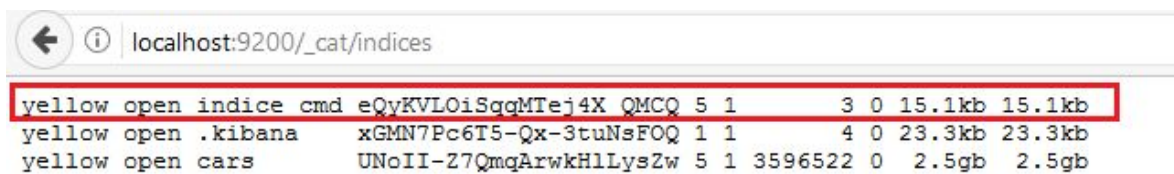
Verificamos que esté corriendo desde: localhost:9200/_cat/indices (aparece ya creado un índice que es la segunda parte del informe)

Ejecutamos logstash con la configuración indicada:

```
Microsoft Windows [Versión 10.0.15063]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

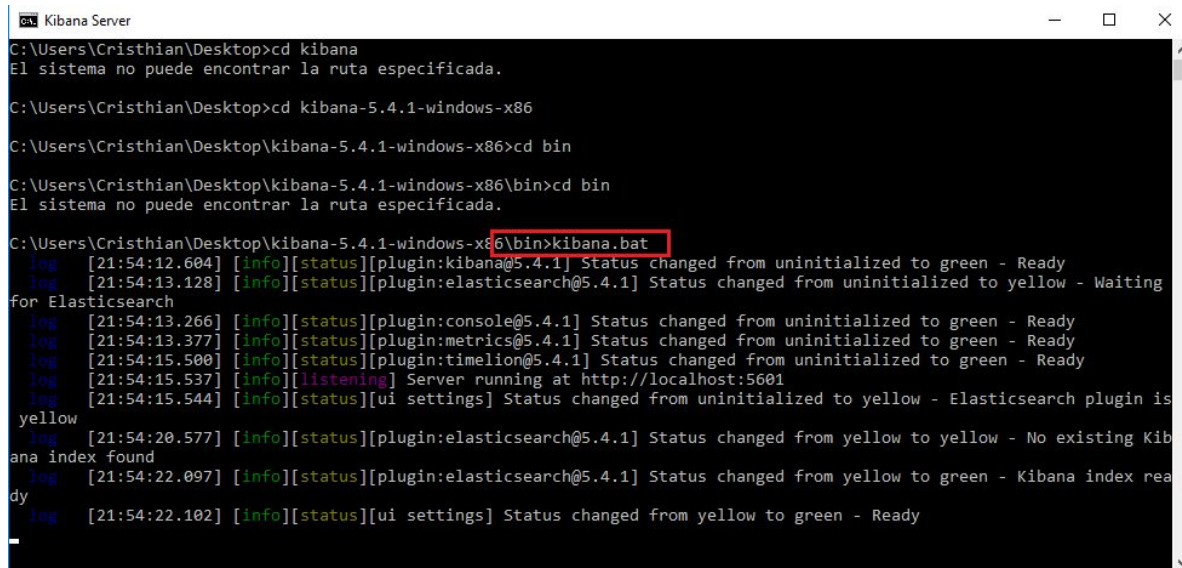
C:\Users\Crsthian>cd Desktop
C:\Users\Crsthian\Desktop>cd logstash-5.4.1
C:\Users\Crsthian\Desktop\logstash-5.4.1>cd bin
C:\Users\Crsthian\Desktop\logstash-5.4.1\bin>logstash -f logs_cmd.conf
Sending Logstash's logs to C:\Users\Crsthian\Desktop\logstash-5.4.1\logs which is now configured via log4j2.properties
[2017-07-19T21:27:45,099][INFO ][logstash.outputs.elasticsearch] Elasticsearch pool URLs updated {:changes=>{:removed=>[], :
added=>[http://localhost:9200/]}}
[2017-07-19T21:27:45,120][INFO ][logstash.outputs.elasticsearch] Running health check to see if an Elasticsearch connection
is working {:healthcheck_url=>http://localhost:9200/, :path=>"/"}
[2017-07-19T21:27:45,570][WARN ][logstash.outputs.elasticsearch] Restored connection to ES instance {:url=>#<URI::HTTP:0x191
6b766 URL:http://localhost:9200/>}
[2017-07-19T21:27:45,574][INFO ][logstash.outputs.elasticsearch] Using mapping template from {:path=>nil}
[2017-07-19T21:27:45,746][INFO ][logstash.outputs.elasticsearch] Attempting to install template {:manage_template=>{"templat
e">{"logstash-*", "version">50001, "settings">{"index.refresh_interval">"5s"}, "mappings">{"_default">{"_type">{"enabl
ed">true, "norms">false}, "dynamic_templates">[{"message_field">{"path_match">"message", "match_mapping_type">"string"
, "mapping">{"type">"text", "norms">false}}, {"string_fields">{"match">"*", "match_mapping_type">"string", "mapping">
{"type">"text", "norms">false, "fields">{"keyword">{"type">"keyword"}}}], "properties">{"@timestamp">{"type">"dat
e", "include_in_all">false}, "@version">{"type">"keyword", "include_in_all">false}, "geoip">{"dynamic">true, "properl
ies">{"ip">{"type">"ip"}, "location">{"type">"geo_point"}, "latitude">{"type">"half_float"}, "longitude">{"type">"ha
lf_float"}}}}}}}}
[2017-07-19T21:27:45,774][INFO ][logstash.outputs.elasticsearch] New Elasticsearch output {:class=>LogStash::Outputs::Elast
icSearch, :hosts=>#<URI::Generic:0x1ee9924e URL://localhost:9200/>}
[2017-07-19T21:27:45,790][INFO ][logstash.pipeline        ] Starting pipeline {"id">"main", "pipeline.workers">4, "pipelin
e.batch.size">125, "pipeline.batch.delay">5, "pipeline.max_inflight">500}
[2017-07-19T21:27:45,967][INFO ][logstash.pipeline        ] Pipeline main started
The stdin plugin is now waiting for input:
mi primer logs[2017-07-19T21:27:46,757][INFO ][logstash.agent ] Successfully started Logstash API endpoint {:port=>
9600}
2017-07-20T01:27:51.179Z CRISTHIAN-PC mi primer log
espero estar generando logs
2017-07-20T01:28:07.640Z CRISTHIAN-PC espero estar generando logs
al parecer todo bien
2017-07-20T01:28:17.229Z CRISTHIAN-PC al parecer todo bien
```

Verificamos que se creó el índice:



index	status	type	size	format	index_size	total_size	primary	replicas	version	wait_time	wait_time_in_millis	wait_time_in_millis
yellow open indice cmd	eQyKVLOiSggMTej4X QMCQ	5	1		3	0	15.1kb	15.1kb				
yellow open .kibana	xGMN7Pc6T5-Qx-3tuNsFOQ	1	1		4	0	23.3kb	23.3kb				
yellow open cars	UNoII-27QmgArwkHlLysZw	5	1		3596522	0	2.5gb	2.5gb				

Ejecutamos kibana con el comando:



```

C:\Users\Cristhian\Desktop>cd kibana
El sistema no puede encontrar la ruta especificada.

C:\Users\Cristhian\Desktop>cd kibana-5.4.1-windows-x86

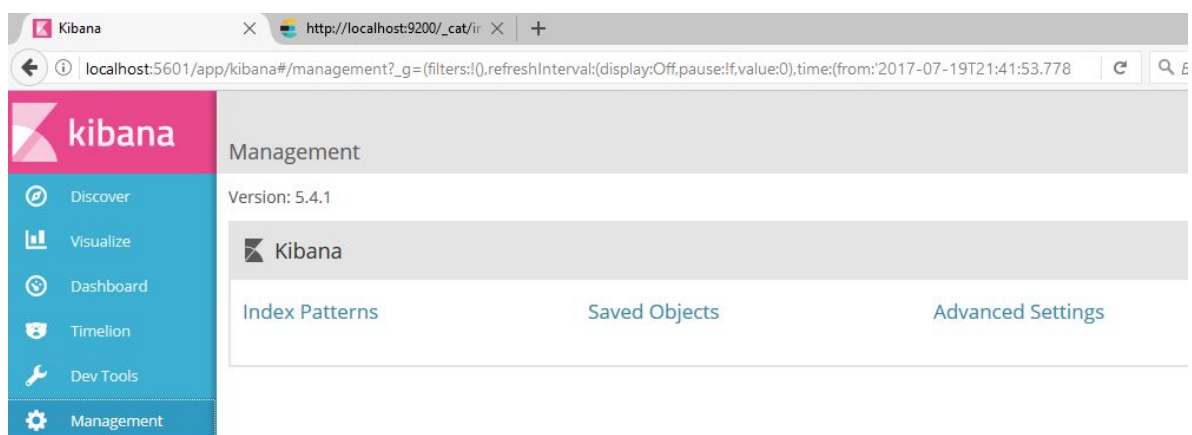
C:\Users\Cristhian\Desktop\kibana-5.4.1-windows-x86>cd bin

C:\Users\Cristhian\Desktop\kibana-5.4.1-windows-x86\bin>cd bin
El sistema no puede encontrar la ruta especificada.

C:\Users\Cristhian\Desktop\kibana-5.4.1-windows-x86\bin>kibana.bat
log [21:54:12.604] [info][status][plugin:kibana@5.4.1] Status changed from uninitialized to green - Ready
log [21:54:13.128] [info][status][plugin:elasticsearch@5.4.1] Status changed from uninitialized to yellow - Waiting
for Elasticsearch
log [21:54:13.266] [info][status][plugin:console@5.4.1] Status changed from uninitialized to green - Ready
log [21:54:13.377] [info][status][plugin:metrics@5.4.1] Status changed from uninitialized to green - Ready
log [21:54:15.500] [info][status][plugin:timelion@5.4.1] Status changed from uninitialized to green - Ready
log [21:54:15.537] [info][listening] Server running at http://localhost:5601
log [21:54:15.544] [info][status][ui settings] Status changed from uninitialized to yellow - Elasticsearch plugin is
yellow
log [21:54:20.577] [info][status][plugin:elasticsearch@5.4.1] Status changed from yellow to yellow - No existing Kib
ana index found
log [21:54:22.097] [info][status][plugin:elasticsearch@5.4.1] Status changed from yellow to green - Kibana index rea
dy
log [21:54:22.102] [info][status][ui settings] Status changed from yellow to green - Ready

```

Para acceder a kibana ocupamos: <http://localhost:5601/app/kibana>



Para visualizar nuestro índice con kibana debemos crear un index patterns

The screenshot shows the Kibana 'Index Patterns' page for the pattern 'indice_cmd'. The left sidebar contains navigation links: Discover, Visualize, Dashboard, Timelion, Dev Tools, and Management. The main content area shows the index pattern 'indice_cmd' with a 'cars' type. Below this, a table lists the fields in the index and their associated core types as recorded by Elasticsearch. The table has columns for name, type, format, searchable, aggregatable, analyzed, excluded, and controls. The fields listed are: @version.keyword (string), message (string), @timestamp (date), message.keyword (string), @version (string), host (string), _source (_source), host.keyword (string), and _id (string).

name	type	format	searchable	aggregatable	analyzed	excluded	controls
@version.keyword	string		✓	✓			[edit]
message	string		✓		✓		[edit]
@timestamp	date		✓	✓			[edit]
message.keyword	string		✓	✓			[edit]
@version	string		✓		✓		[edit]
host	string		✓		✓		[edit]
_source	_source						[edit]
host.keyword	string		✓	✓			[edit]
_id	string						[edit]

En discover podemos ver un gráfico con la actividad de los logs, ver los campos que tiene cada uno de ellos, dentro de eso encontramos los mensajes ingresados por consola.

The screenshot shows the Kibana 'Discover' page. The left sidebar contains navigation links: Discover, Visualize, Dashboard, Timelion, Dev Tools, and Management. The main content area shows a histogram of log activity over time. The x-axis is labeled '@timestamp per minute' and the y-axis is labeled 'Count'. The histogram shows a peak in activity around 21:28. Below the histogram, a table lists the log messages. The messages are: 'mi primer log', 'espero estar generando ...', and 'al parecer todo bien'. Each message has a 'color_slug' field and a 'timestamp' field.

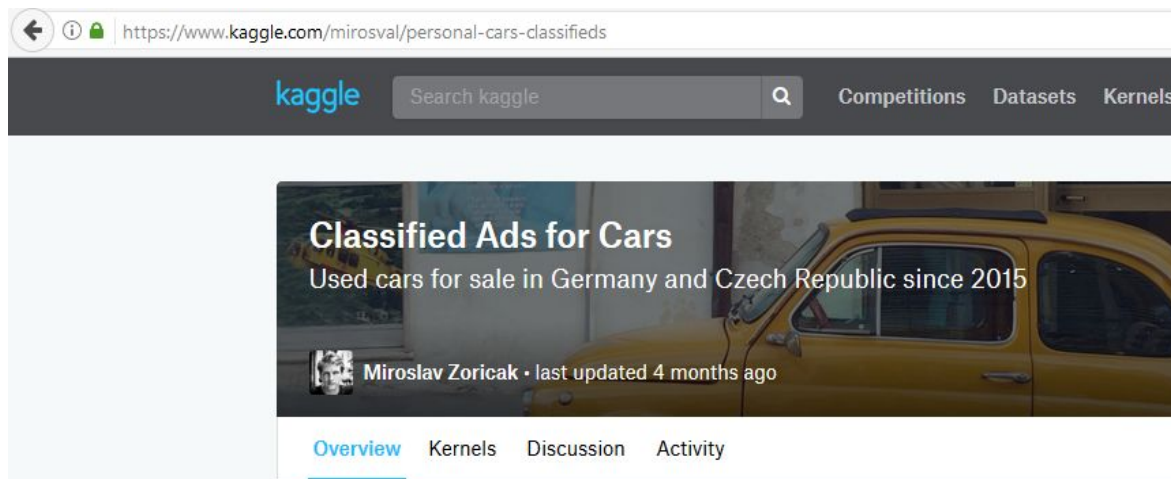
Time	color_slug
July 19th 2017, 21:28:17.229	-
July 19th 2017, 21:28:07.640	-
July 19th 2017, 21:27:51.179	-

4.2. Indexar y visualizar logs obtenidos desde un archivo .csv a través de logstash

Lo primero es buscar un archivo que queramos utilizar, encontré una página con datos gratuitos.

<https://www.kaggle.com>

Opté por un dataset en versión csv con información detallada de los autos vendidos en Alemania y República Checa.



Se descarga el csv y se descomprime en el escritorio.

Siguiendo la línea de la primera parte del informe, creamos un archivo de configuración en la carpeta bin de logstash para comenzar:

```
logs_cars.conf
1 input{
2   file{
3     path => "C:\Users\Cristhian\Desktop\cars.csv"
4     start_position => "beginning"
5     sincedb => "/dev/null"
6   }
7 }
8 filter {
9   csv{
10    separator => ","
11    columns => [ "maker", "model", "mileage", "manufacture_year", "engine_displacement", "engine_power",
12               "body_type", "color_slug", "stk_year", "transmission", "door_count", "seat_count", "fuel_type",
13               "date_created", "date_last_seen", "price_eur" ]
14  }
15  mutate{convert => ["mileage", "integer"]}
16  mutate{convert => ["price_eur", "float"]}
17  mutate{convert => ["engine_power", "integer"]}
18  mutate{convert => ["door_count", "integer"]}
19  mutate{convert => ["seat_count", "integer"]}
20 }
21 output{
22   elasticsearch{
23     hosts => "localhost"
24     index => "cars"
25     document_type => "sold_cars"
26   }
27   stdout{}
28 }
```

En input se debe especificar la ruta del archivo csv del que queremos obtener los datos.

En filter se debe describir como es el formato del csv a utilizar, en este caso se toma como delimitador la coma “,” y se indica cuáles son las columnas que contiene, además se incluye un if para identificar el registro con el nombre de las columnas para no considerarlo como dato.

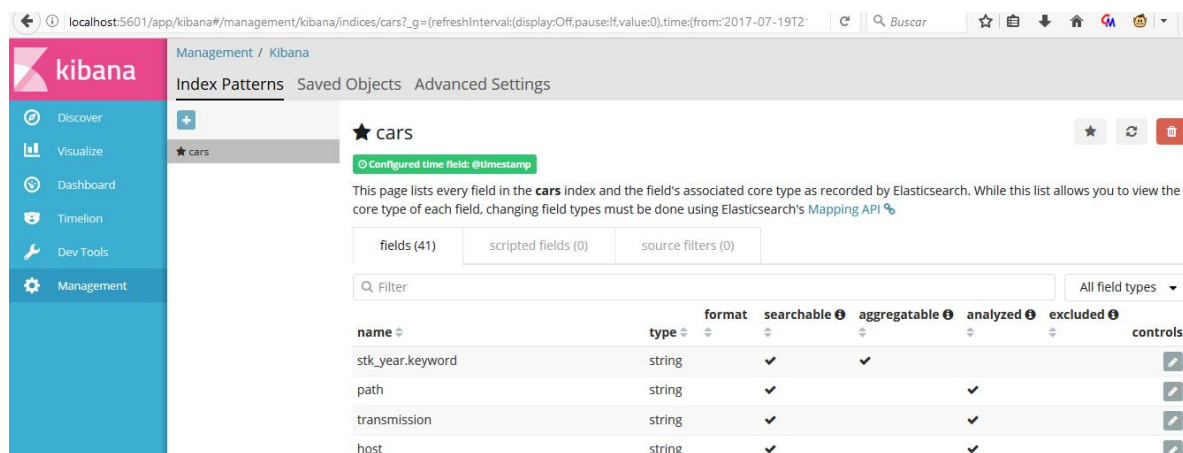
Además se especifican de qué tipo son los datos según su columna con “mutate”

Finalmente se especifica en output el host y el nombre del índice que será alojado en elastic.

Ejecutamos elasticsearch , logstash con la configuración indicada

Ejecutamos kibana

Para visualizar con kibana debemos crear un index patterns en la ventana “Management”



Management / Kibana

Index Patterns Saved Objects Advanced Settings

★ cars

Configured time field: @timestamp

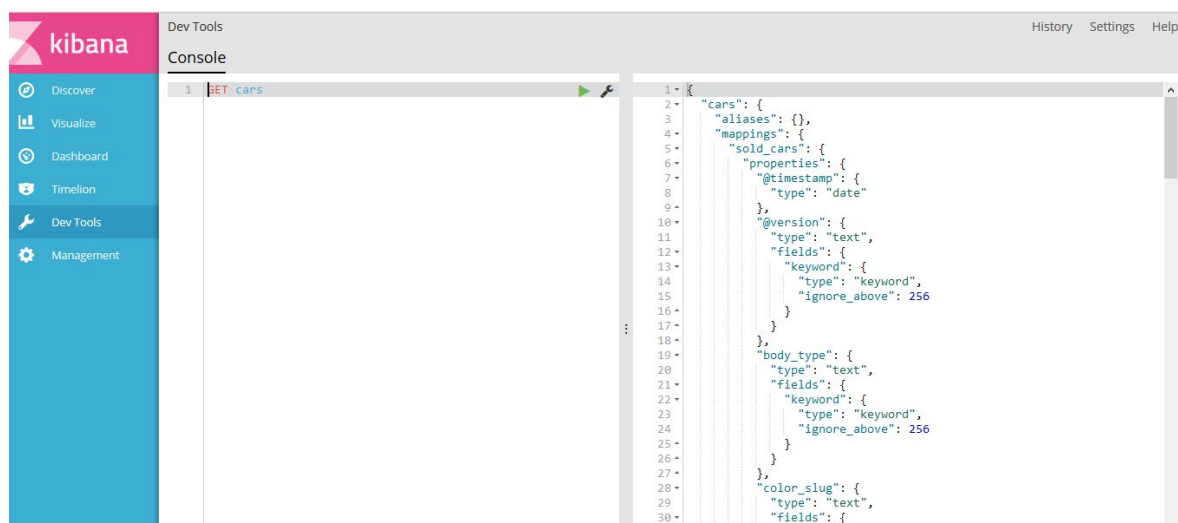
This page lists every field in the **cars** index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's [Mapping API](#).

fields (41) scripted fields (0) source filters (0)

Filter

name	type	format	searchable	aggregatable	analyzed	excluded	controls
stk_year.keyword	string		✓	✓			
path	string		✓		✓		
transmission	string		✓		✓		
host	string		✓		✓		

Kibana nos permite ejecutar comandos para realizar distintas operaciones sobre el índice en la ventana “Dev Tools”, por ejemplo podemos ver su estructura:



Dev Tools

Console

```

1 GET cars
2
3 {
4   "cars": {
5     "aliases": {},
6     "mappings": {
7       "sold_cars": {
8         "properties": {
9           "@timestamp": {
10            "type": "date"
11          },
12          "@version": {
13            "type": "text",
14            "fields": {
15              "keyword": {
16                "type": "keyword",
17                "ignore_above": 256
18              }
19            }
20          },
21          "body_type": {
22            "type": "text",
23            "fields": {
24              "keyword": {
25                "type": "keyword",
26                "ignore_above": 256
27              }
28            }
29          },
30          "color_slug": {
31            "type": "text",
32            "fields": {

```

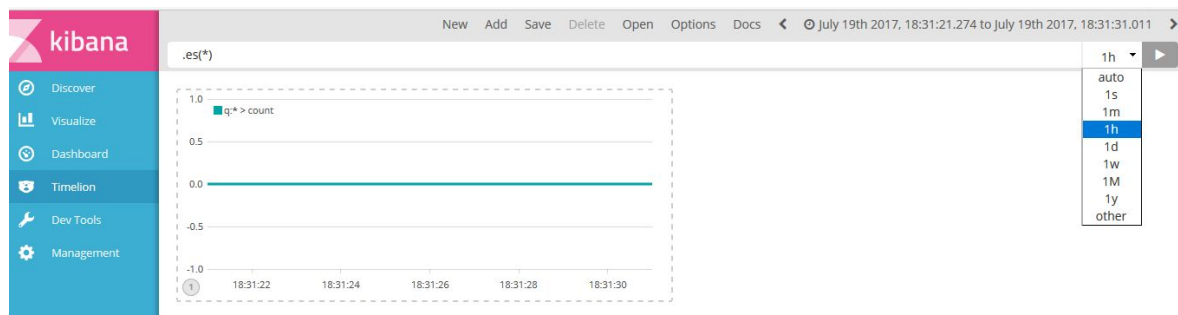
O contar la cantidad de registros

Console

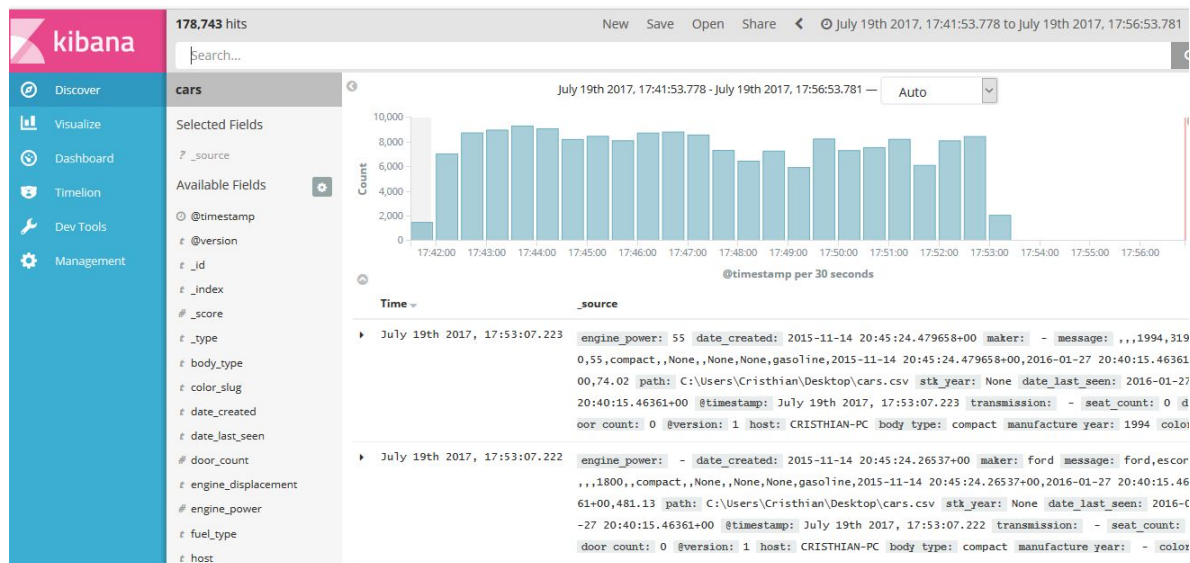
```
1 get /cars/_count
```

```
1 {
2   "count": 3596522,
3   "_shards": {
4     "total": 5,
5     "successful": 5,
6     "failed": 0
7   }
8 }
```

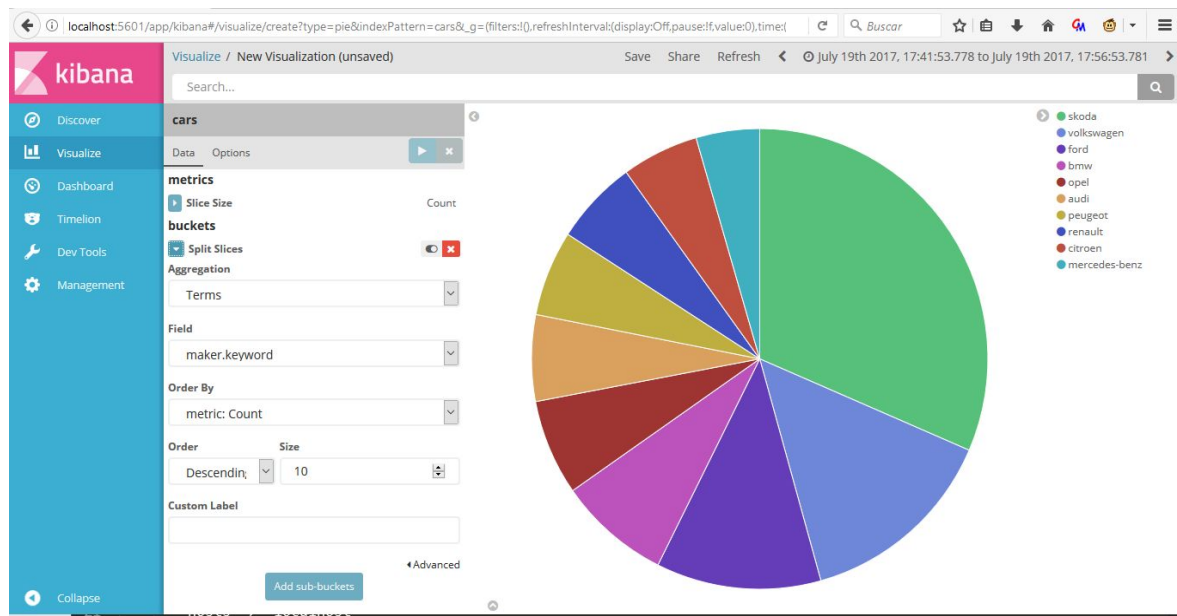
En “TimeLion” se puede ver la cantidad de logs generados en un periodo de tiempo determinado.



En discover podemos ver un gráfico con la actividad de los logs, ver los campos que tiene cada uno de ellos.



En ventana “Visualize” se pueden realizar distintos tipos de gráficos que permiten interpretar la información según nuestra necesidad, en este caso podemos ver las 10 marcas de autos más vendidas:

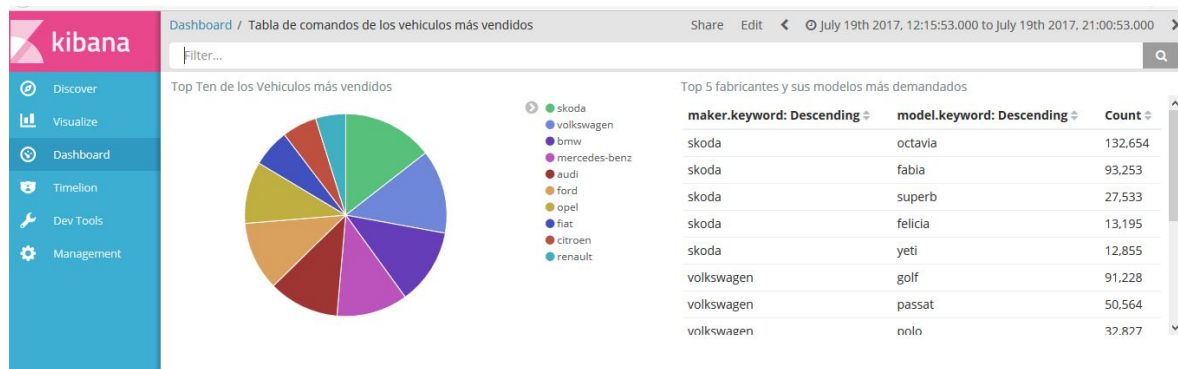


O generar una visualización tipo tabla, en donde se filtra por tipo de marca y como subcaracterística los distintos modelos de las marcas de los vehículos vendidos.

The image shows a Kibana table visualization titled 'cars'. The table displays the distribution of car models by manufacturer, sorted by 'maker.keyword: Descending' and 'model.keyword: Descending'. The table has three columns: 'maker.keyword: Descending', 'model.keyword: Descending', and 'Count'. The data is as follows:

maker.keyword: Descending	model.keyword: Descending	Count
skoda	octavia	16,766
skoda	fabia	11,032
skoda	superb	3,177
skoda	felicia	2,043
skoda	rapid	1,234
volkswagen	golf	4,797
volkswagen	passat	4,497
volkswagen	touran	1,382
volkswagen	polo	1,166
volkswagen	sharan	909

La ventana “Dashboard” es una tabla de control donde se puede utilizar los graficos creados para ordenarlos y crear un resumen de lo que nos interesa saber de la información que queremos analizar.



4.3. Indexar y visualizar logs obtenidos desde un .csv generado desde apache jmeter al realizar una petición http a un sitio web

Apache Jmeter es un software de código libre hecha en java para ver el comportamiento funcional y medir el rendimiento, está orientado a la realización de pruebas. En este caso se realizó una prueba básica a la plataforma de la universidad (<http://Phoenix.cic.userena.cl>).

Se descarga apache jmeter desde la página <http://jmeter.apache.org/>

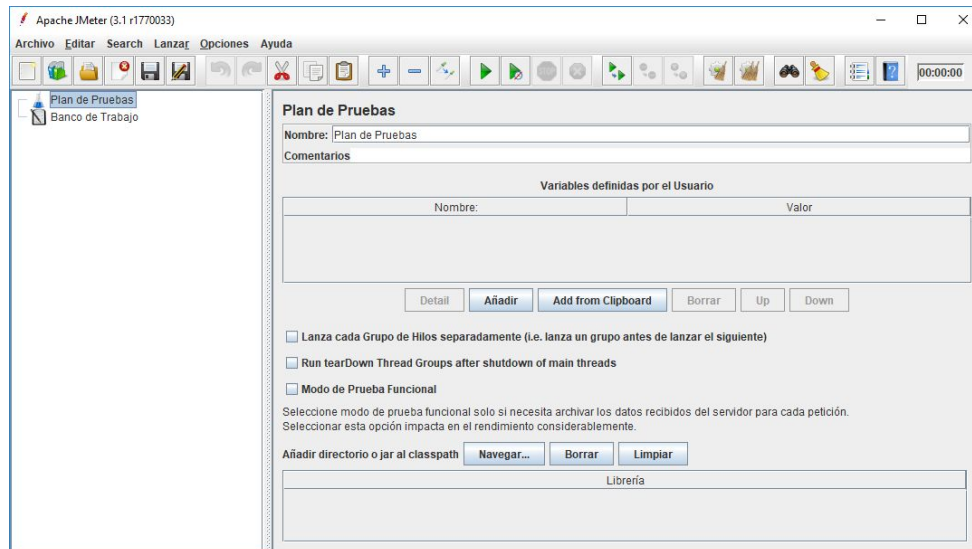


Descomprimir en el escritorio.

Se ejecuta jmeter desde cmd:

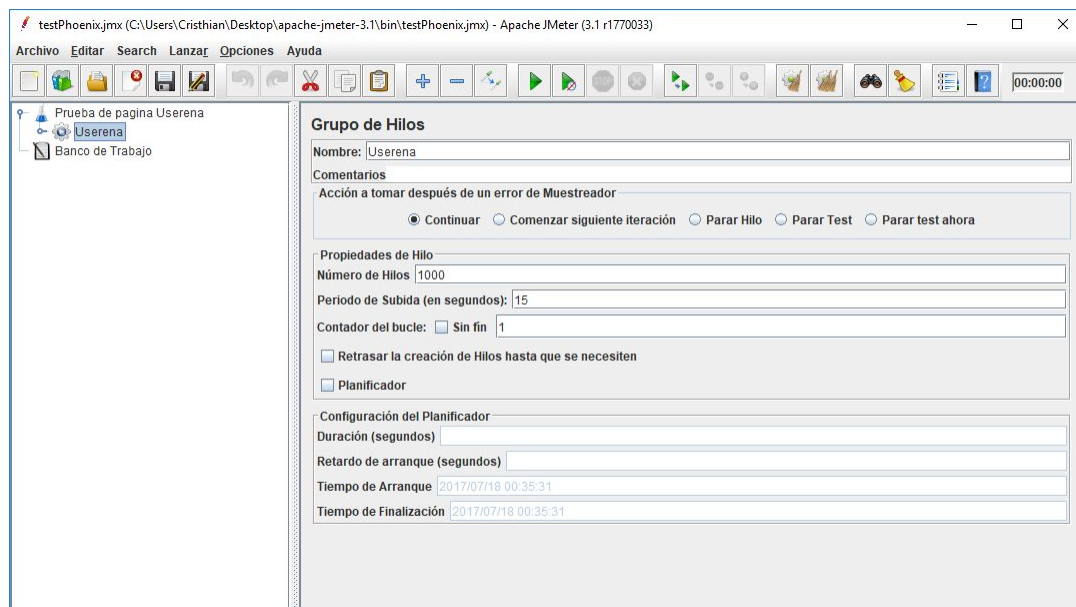
```
C:\Users\Cristhian\Desktop\apache-jmeter-3.1\bin>jmeter.bat
Writing log file to: C:\Users\Cristhian\Desktop\apache-jmeter-3.1\bin\jmeter.log
=====
Don't use GUI mode for load testing, only for Test creation and Test debugging !
For load testing, use NON GUI Mode & adapt Java Heap to your test requirements
=====
jul 19, 2017 4:33:40 PM java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0x80000002. Windows RegCreateKeyEx(...)
returned error code 5.
```

Y se muestra la interfaz principal:

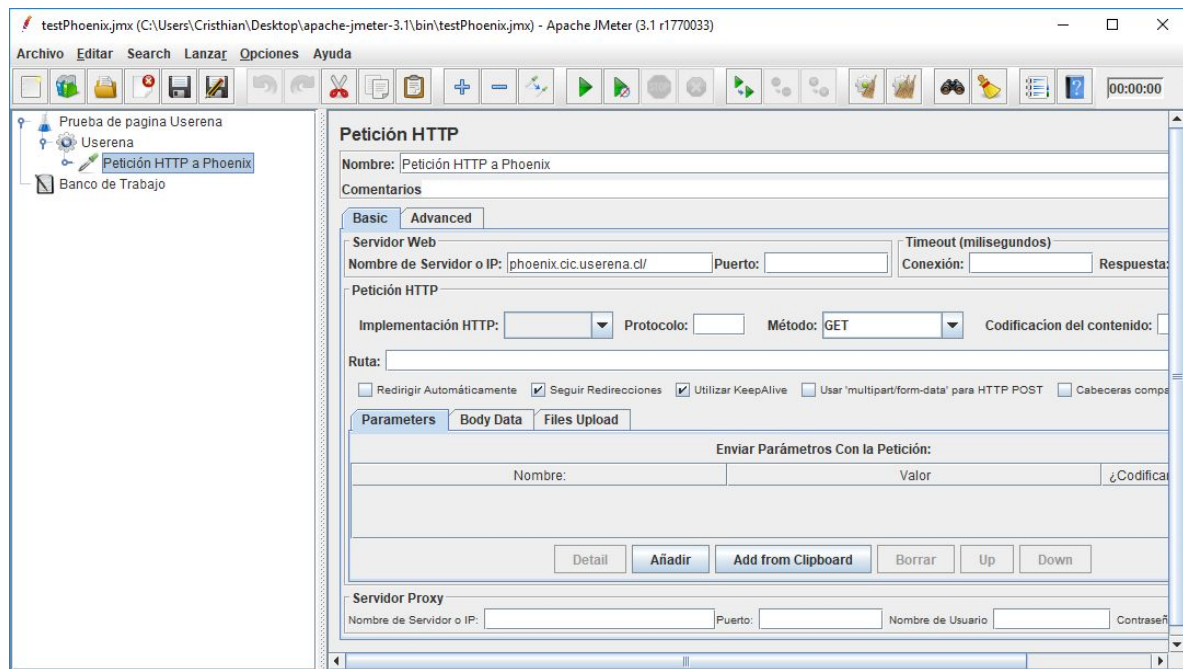


Para crear una prueba creamos un grupo de hilos donde cada hilo representa un usuario, el grupo de hilos se llamará Userena.

Se da la cantidad de hilos(usuarios) a considerar y el tiempo determinado de la prueba.



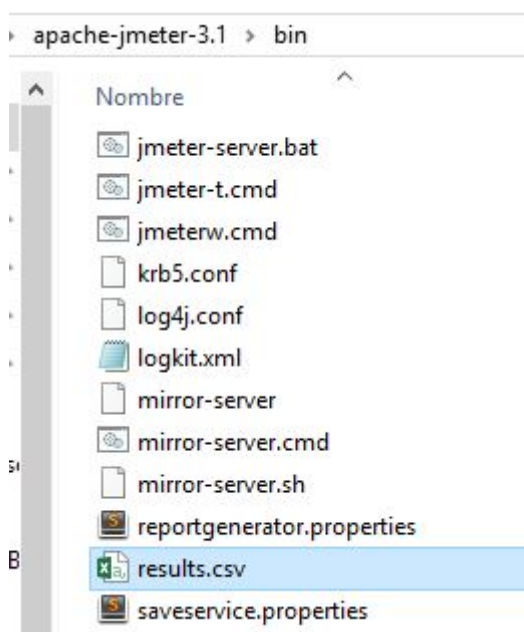
Se crea una petición HTTP donde se debe identificar el sitio.



Guardamos esta prueba y la ejecutamos por consola:

```
C:\Users\Cristhian\Desktop\apache-jmeter-3.1\bin>jmeter -n -t testPhoenix.jmx -l results.csv
```

Y en la misma carpeta bin se genera el archivo csv con la información:



	A	B	C	D	E	F	G	H
1	timeStamp,elapsed,label,responseCode,responseMessage,threadName,dataType,success,failureMessage,							
2	1500409754550,7737,Petición HTTP a Phoenix,200,OK,Userena 1-178,text,true,,9191,382,603,603,362,0,177							
3	1500409752627,9664,Petición HTTP a Phoenix,200,OK,Userena 1-62,text,true,,9191,382,603,603,619,0,99							
4	1500409754536,7753,Petición HTTP a Phoenix,200,OK,Userena 1-177,text,true,,9191,382,603,603,376,0,191							
5	1500409753416,8915,Petición HTTP a Phoenix,200,OK,Userena 1-97,text,true,,9191,382,604,604,235,0,165							
6	1500409755371,6978,Petición HTTP a Phoenix,200,OK,Userena 1-227,text,true,,9191,382,604,604,336,0,93							
7	1500409754228,8121,Petición HTTP a Phoenix,200,OK,Userena 1-156,text,true,,9191,382,604,604,195,0,90							
8	1500409754328,8163,Petición HTTP a Phoenix,200,OK,Userena 1-163,text,true,,9191,382,611,611,166,0,93							
9	1500409752780,10168,Petición HTTP a Phoenix,200,OK,Userena 1-67,text,true,,9191,382,611,611,552,0,459							

Teniendo nuestro csv, podemos configurar logstash para poder obtener los datos y enviarlos a elasticsearch como en la segunda parte de la sección.

El archivo de configuración es el siguiente:

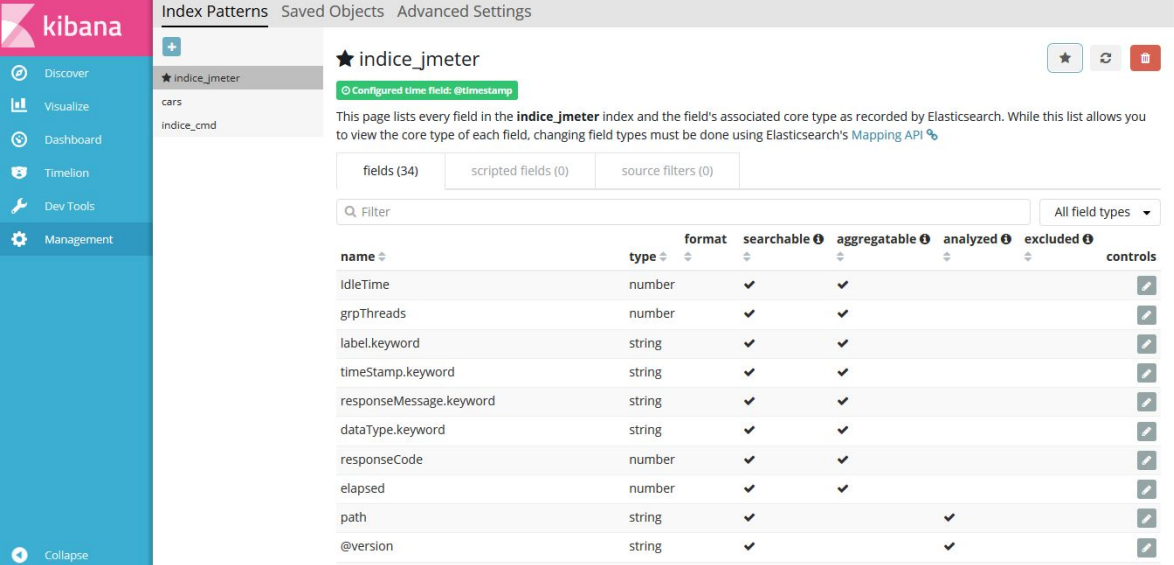
```
logs_csv.conf
input{
  file{
    path => "C:\Users\Cristhian\Desktop\datos_phoenix.csv"
    start_position => "beginning"
  }
}
filter{
  csv{
    separator => ","
    columns => ["timeStamp", "elapsed", "label", "responseCode", "responseMessage", "threadName", "dataType", "success", "failureMessage",
    "bytes", "sentBytes", "grpThreads", "allThreads", "Latency", "IdleTime", "Connect"]
  }
  mutate{convert => ["timeStamp","string"]}
  mutate{convert => ["elapsed","float"]}
  mutate{convert => ["label","string"]}
  mutate{convert => ["responseCode","integer"]}
  mutate{convert => ["responseMessage","string"]}
  mutate{convert => ["threadName","string"]}
  mutate{convert => ["success","boolean"]}
  mutate{convert => ["failureMessage","string"]}
  mutate{convert => ["bytes","float"]}
  mutate{convert => ["sentBytes","float"]}
  mutate{convert => ["grpThreads","float"]}
  mutate{convert => ["allThreads","float"]}
  mutate{convert => ["Latency","float"]}
  mutate{convert => ["IdleTime","integer"]}
  mutate{convert => ["Connect","integer"]}
}
output{
  elasticsearch{
    hosts => "localhost"
    index => "indice_jmeter"
  }
  stdout{}
}
```

Nuevamente en input se entrega la dirección del archivo csv

En filter se especifican las columnas correspondientes a los datos existentes pero antes existe un if que está encargado de encontrar alguna tupla con un mensaje similar al nombre de una columna en específico, dado que si se nombre es porque estamos en el registro de columnas y no son partes de los datos por lo que no se debe considerar.

Finalmente en output definimos el host desde donde se corre elasticsearch y el nombre del índice.

El dato en el que se quiere basar la información es la latencia de cada hilo al momento de hacer la petición http.



kibana Index Patterns Saved Objects Advanced Settings

★ indice_jmeter

cars
indice_cmd

★ indice_jmeter

Configured time field: @timestamp

This page lists every field in the **indice_jmeter** index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's [Mapping API](#).

fields (34) scripted fields (0) source filters (0)

Filter

All field types

name	type	format	searchable	aggregatable	analyzed	excluded	controls
IdleTime	number		✓	✓			
grpThreads	number		✓	✓			
label.keyword	string		✓	✓			
timeStamp.keyword	string		✓	✓			
responseMessage.keyword	string		✓	✓			
dataType.keyword	string		✓	✓			
responseCode	number		✓	✓			
elapsed	number		✓	✓			
path	string		✓		✓		
@version	string		✓		✓		

Collapse

5. Conclusión

Solr es una poderosa herramienta para realizar indexaciones y búsquedas. La flexibilidad que posee para trabajar con diferentes formatos es la característica clave (a parte de las capacidades de la búsqueda), dado a que se puede acomodar a las necesidades de los usuarios. Las funcionalidades que ofrece facilita la construcción de aplicaciones mediante las APIs disponibles. En este documento solo se exploró con la API de Java (SolrJ) pero de forma oficial Solr cuenta con APIs para Java, Javascript, Python, y Ruby. Las APIs ofrecen métodos simples de alto nivel para comunicarse con Solr, estos métodos son los que hacen posible la realización de solicitudes como indexaciones y búsqueda. La complejidad desaparece ante los ojos del usuario, no solo con las APIs sino de forma general, ya sea mediante comandos o la misma interfaz de usuario. Solr está construido de forma tal que no es necesario saber cómo funciona internamente en la indexación y búsqueda de archivos, todos estos detalles se esconden al usuario, pero esto no implica que no se pueda personalizar. En la indexación por base de datos se pudo ver como se puede trabajar con el motor que se desee y como se puede modificar el esquema de los documentos a indexar, además de trabajar de forma distribuido Solr con SolrCloud como alternativa. En definitiva Solr es una muy buena opción a utilizar dado a que se construye en base a las necesidades que pueden llegar a tener los usuarios, gracias a la flexibilidad que ofrece.

Por su parte Elasticsearch trabaja de forma distribuida (o pensado para casos en los que se requiera trabajar distribuidamente) por defecto. La gran diferencia con Solr, es la capacidad de obtener la relevancia de los resultados obtenidos al realizar una búsqueda. Esto permite realizar análisis y filtros de mayor nivel, ya que no solo se obtiene lo que se está buscando en la consulta, sino además, saber que tan importante es ese resultado respecto a lo buscado. Otro factor de Elasticsearch, es la forma en que está construido, dado a que permite enlazarlo a logstash y kibana, obteniendo un amplio abanico de características para realizar análisis. Logstash soporta una variedad de entradas que extraen eventos de una multitud de fuentes comunes, todo al mismo tiempo. De esta forma se puede ingresar datos fácilmente de registros, métricas, aplicaciones web, almacenes de datos y varios servicios, todos de manera continua y en streaming. Y finalmente Kibana permite visualizar datos de Elasticsearch a través de histogramas, gráficos de líneas, gráficos circulares, sunbursts, y mucho más, aprovechando las capacidades completas de Elasticsearch.