EBook Gratis

APRENDIZAJE Elasticsearch

Free unaffiliated eBook created from
Stack Overflow contributors.

#elasticsear

ch

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: elasticsearch

It is an unofficial and free Elasticsearch ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Elasticsearch.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Elasticsearch

Observaciones

Elasticsearch es un servidor de búsqueda de código abierto avanzado basado en Lucene y escrito en Java.

Proporciona funciones de búsqueda distribuidas de texto completo y parcial, basadas en consultas y geolocalización accesibles a través de una API REST HTTP.

Versiones

Versión	Fecha de lanzamiento
5.2.1	2017-02-14
5.2.0	2017-01-31
5.1.2	2017-01-12
5.1.1	2016-12-08
5.0.2	2016-11-29
5.0.1	2016-11-15
5.0.0	2016-10-26
2.4.0	2016-08-31
2.3.0	2016-03-30
2.2.0	2016-02-02
2.1.0	2015-11-24
2.0.0	2015-10-28
1.7.0	2015-07-16
1.6.0	2015-06-09
1.5.0	2015-03-06
1.4.0	2014-11-05
1.3.0	2014-07-23

Versión	Fecha de lanzamiento
1.2.0	2014-05-22
1.1.0	2014-03-25
1.0.0	2014-02-14

Examples

Instalando Elasticsearch en Ubuntu 14.04

Prerrequisitos

Para ejecutar Elasticsearch, se requiere un Java Runtime Environment (JRE) en la máquina. Elasticsearch requiere Java 7 o superior y recomienda `Oracle JDK version 1.8.0_73` .

Instalar Oracle Java 8

```
sudo add-apt-repository -y ppa:webupd8team/java
sudo apt-get update
echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" | sudo debconf-
set-selections
sudo apt-get install -y oracle-java8-installer
```

Comprobar la versión de Java

```
java -version
```

Descargar e instalar el paquete

Utilizando binarios

1. Descarga la última versión estable de Elasticsearch aquí .
2. Descomprima el archivo y ejecute

Linux:

```
$ bin/elasticsearch
```

Utilizando apt-get

Una alternativa a la descarga de elasticsearch desde el sitio web es instalarlo, usando `apt-get` .

```
wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
echo "deb https://packages.elastic.co/elasticsearch/2.x/debian stable main" | sudo tee -a
```

```
/etc/apt/sources.list.d/elasticsearch-2.x.list
sudo apt-get update && sudo apt-get install elasticsearch
sudo /etc/init.d/elasticsearch start
```

Instalación de elasticsearch versión 5.x

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
sudo apt-get install apt-transport-https
echo "deb https://artifacts.elastic.co/packages/5.x/apt stable main" | sudo tee -a
/etc/apt/sources.list.d/elastic-5.x.list
sudo apt-get update && sudo apt-get install elasticsearch
```

Ejecutando como un servicio en Linux:

Después de instalar lo anterior no se inicia solo. por lo que tenemos que iniciarlo como un servicio. Cómo iniciar o detener Elasticsearch depende de si su sistema utiliza SysV init o systemd. Puedes comprobarlo con el siguiente comando.

```
ps -p 1
```

Si su distribución utiliza SysV init, deberá ejecutar:

```
sudo update-rc.d elasticsearch defaults 95 10
sudo /etc/init.d/elasticsearch start
```

De lo contrario, si su distribución está usando systemd:

```
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch.service
```

Ejecute el comando `CURL` desde su navegador o un cliente REST para verificar si Elasticsearch se ha instalado correctamente.

```
curl -X GET http://localhost:9200/
```

Instalación de Elasticsearch en Windows

Prerrequisitos

La versión para Windows de Elasticsearch se puede obtener en este enlace: https://www.elastic.co/downloads/elasticsearch . La última versión estable está siempre en la parte superior.

Como estamos instalando en Windows, necesitamos el archivo `.ZIP` . Haga clic en el enlace en la sección `Downloads:` y guarde el archivo en su computadora.

Esta versión de elastic es "portátil", lo que significa que no necesita ejecutar un instalador para usar el programa. Descomprima el contenido del archivo en una ubicación que pueda recordar fácilmente. Para la demostración, asumiremos que has descomprimido todo en `C:\elasticsearch` .

Tenga en cuenta que el archivo contiene una carpeta llamada `elasticsearch-<version>` de forma predeterminada, puede extraer esa carpeta a `C:\` y cambiarle el nombre a `elasticsearch` o crear `C:\elasticsearch` usted mismo, luego descomprimir solo el *contenido* de la carpeta en el archivo a ahi

Dado que Elasticsearch está escrito en Java, necesita el entorno de ejecución de Java para funcionar. Entonces, antes de ejecutar el servidor, compruebe si Java está disponible abriendo un símbolo del sistema y escribiendo:

```
java -version
```

Deberías obtener una respuesta como esta:

```
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) Client VM (build 25.91-b14, mixed mode)
```

Si ves lo siguiente en su lugar

> 'java' no se reconoce como un comando interno o externo, un programa operable o un archivo por lotes.

Java no está instalado en su sistema o no está configurado correctamente. Puedes seguir este tutorial para (re) instalar Java. Además, asegúrese de que estas variables de entorno estén configuradas en valores similares:

Variable	Valor
JAVA_HOME	C: \ Archivos de programa \ Java \ jre
CAMINO	...; C: \ Archivos de programa \ Java \ jre

Si aún no sabe cómo inspeccionar estas variables, consulte este tutorial .

Ejecutar desde archivo por lotes

Con Java instalado, abre la carpeta `bin` . Se puede encontrar directamente en la carpeta en la que se descomprimió todo, por lo que debería estar en `c:\elasticsearch\bin` . Dentro de esta carpeta hay un archivo llamado `elasticsearch.bat` que se puede usar para iniciar Elasticsearch en una ventana de comandos. Esto significa que la información registrada por el proceso será visible en la ventana del símbolo del sistema. Para detener el servidor, presione `CTRL C` o simplemente cierre la ventana.

Ejecutar como un servicio de Windows

Lo ideal es que no desee tener una ventana adicional de la que no pueda deshacerse durante el desarrollo y, por este motivo, Elasticsearch puede configurarse para que se ejecute como un servicio.

Antes de poder instalar Elasticsearch como un servicio, necesitamos agregar una línea al archivo `C:\elasticsearch\config\jvm.options` :

> El instalador del servicio requiere que la configuración de tamaño de pila de subprocesos esté configurada en `jvm.options` antes de instalar el servicio. En Windows de 32 bits, debe agregar `-Xss320k` [...] y en Windows de 64 bits, debe agregar `-Xss1m` al archivo `jvm.options` . [fuente]

Una vez que haya realizado ese cambio, abra un símbolo del sistema y navegue hasta el directorio `bin` ejecutando el siguiente comando:

```
C:\Users\user> cd c:\elasticsearch\bin
```

La gestión del servicio es manejada por `elasticsearch-service.bat` . En versiones anteriores, este archivo podría llamarse simplemente `service.bat` . Para ver todos los argumentos disponibles, ejecútalo sin ninguna:

```
C:\elasticsearch\bin> elasticsearch-service.bat

Usage: elasticsearch-service.bat install|remove|start|stop|manager [SERVICE_ID]
```

La salida también nos dice que hay un argumento `SERVICE_ID` opcional, pero podemos ignorarlo por ahora. Para instalar el servicio, simplemente ejecute:

```
C:\elasticsearch\bin> elasticsearch-service.bat install
```

Después de instalar el servicio, puede iniciarlo y detenerlo con los argumentos correspondientes. Para iniciar el servicio, ejecute

```
C:\elasticsearch\bin> elasticsearch-service.bat start
```

y para detenerlo, correr

```
C:\elasticsearch\bin> elasticsearch-service.bat stop
```

Si prefiere una GUI para administrar el servicio, puede usar el siguiente comando:

```
C:\elasticsearch\bin> elasticsearch-service.bat manager
```

Esto abrirá Elastic Service Manager, que le permite personalizar algunas configuraciones

relacionadas con el servicio, así como detener / iniciar el servicio utilizando los botones que se encuentran en la parte inferior de la primera pestaña.

Indexación y recuperación de un documento

Se accede a Elasticsearch a través de una API REST HTTP, generalmente utilizando la biblioteca cURL. Los mensajes entre el servidor de búsqueda y el cliente (su o su aplicación) se envían en forma de cadenas JSON. Por defecto, Elasticsearch se ejecuta en el puerto 9200.

En los ejemplos a continuación, se agrega `?pretty` Pretty para decirle a Elasticsearch que pretenda la respuesta JSON. Cuando utilice estos puntos finales dentro de una aplicación, no necesita agregar este parámetro de consulta.

Indexación de documentos

Si pretendemos actualizar la información dentro de un índice más adelante, es una buena idea asignar identificaciones únicas a los documentos que indexamos. Para agregar un documento al índice llamado `megacorp` , con el tipo `employee` y la ejecución de ID `1` :

```
curl -XPUT "http://localhost:9200/megacorp/employee/1?pretty" -d'
{
    "first_name" : "John",
    "last_name" :  "Smith",
    "age" :        25,
    "about" :      "I love to go rock climbing",
    "interests": [ "sports", "music" ]
}'
```

Respuesta:

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "1",
  "_version": 1,
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "created": true
}
```

El índice se crea si no existe cuando enviamos la llamada PUT.

Indexación sin identificación

```
POST /megacorp/employee?pretty
{
```

```
    "first_name" :  "Jane",
    "last_name" :   "Smith",
    "age" :         32,
    "about" :       "I like to collect rock albums",
    "interests":  [ "music" ]
}
```

Respuesta:

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "AVYg2mBJYy9ijdngfeGa",
  "_version": 1,
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "created": true
}
```

Recuperando documentos

```
curl -XGET "http://localhost:9200/megacorp/employee/1?pretty"
```

Respuesta:

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "1",
  "_version": 1,
  "found": true,
  "_source": {
    "first_name": "John",
    "last_name": "Smith",
    "age": 25,
    "about": "I love to go rock climbing",
    "interests": [
      "sports",
      "music"
    ]
  }
}
```

Obtenga 10 documentos del índice de `megacorp` con el `employee` tipo:

```
curl -XGET "http://localhost:9200/megacorp/employee/_search?pretty"
```

Respuesta:

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 2,
    "max_score": 1,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "1",
        "_score": 1,
        "_source": {
          "first_name": "John",
          "last_name": "Smith",
          "age": 25,
          "about": "I love to go rock climbing",
          "interests": [
            "sports",
            "music"
          ]
        }
      },
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "AVYg2mBJYy9ijdngfeGa",
        "_score": 1,
        "_source": {
          "first_name": "Jane",
          "last_name": "Smith",
          "age": 32,
          "about": "I like to collect rock albums",
          "interests": [
            "music"
          ]
        }
      }
    ]
  }
}
```

Búsqueda simple utilizando la consulta de `match` , que busca coincidencias exactas en el campo provisto:

```
curl -XGET "http://localhost:9200/megacorp/employee/_search" -d'
{
    "query" : {
        "match" : {
            "last_name" : "Smith"
        }
    }
}'
```

Respuesta:

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 0.6931472,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "1",
        "_score": 0.6931472,
        "_source": {
          "first_name": "John",
          "last_name": "Smith",
          "age": 25,
          "about": "I love to go rock climbing",
          "interests": [
            "sports",
            "music"
          ]
        }
      }
    ]
  }
}
```

Parámetros básicos de búsqueda con ejemplos:

De forma predeterminada, el documento indexado completo se devuelve como parte de todas las búsquedas. Esto se conoce como la fuente (campo `_source` en los resultados de búsqueda). Si no queremos que se devuelva todo el documento de origen, podemos solicitar que solo se devuelvan algunos campos dentro de la fuente, o podemos establecer `_source` en falso para omitir el campo por completo.

Este ejemplo muestra cómo devolver dos campos, `account_number` y `balance` (dentro de `_source`), de la búsqueda:

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match_all": {} },
  "_source": ["account_number", "balance"]
}'
```

Tenga en cuenta que el ejemplo anterior simplemente reduce la información devuelta en el campo `_source` . Todavía devolverá solo un campo llamado `_source` pero solo se `account_number` los campos `account_number` y `balance` .

Si proviene de un fondo SQL, el concepto anterior es algo similar a la consulta SQL

```
SELECT account_number, balance FROM bank;
```

Ahora vamos a pasar a la parte de consulta. Anteriormente, hemos visto cómo se `match_all` consulta `match_all` para hacer coincidir todos los documentos. Ahora introduzcamos una nueva consulta llamada consulta de coincidencia, que se puede considerar como una consulta de búsqueda de campo básica (es decir, una búsqueda realizada en un campo específico o conjunto de campos).

Este ejemplo devuelve la cuenta con el `account_number` establecido en `20` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "account_number": 20 } }
}'
```

Este ejemplo devuelve todas las cuentas que contienen el término "molino" en la `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "address": "mill" } }
}'
```

Este ejemplo devuelve todas las cuentas que contienen el término "molino" o "carril" en la `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "address": "mill lane" } }
}'
```

Este ejemplo es una variante de `match` (`match_phrase`) que divide la consulta en términos y solo devuelve documentos que contienen todos los términos en la `address` en las mismas posiciones entre sí [1] .

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match_phrase": { "address": "mill lane" } }
}'
```

Ahora vamos a introducir la consulta bool (ean). La consulta bool nos permite componer consultas más pequeñas en consultas más grandes usando la lógica booleana.

Este ejemplo compone dos consultas de coincidencia y devuelve todas las cuentas que contienen "mill" y "lane" en la dirección:

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
```

```
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

En el ejemplo anterior, la cláusula bool `must` especificar todas las consultas que deben ser verdaderas para que un documento se considere una coincidencia.

En contraste, este ejemplo compone dos consultas de coincidencia y devuelve todas las cuentas que contienen "mill" o "carril" en la `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "should": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

En el ejemplo anterior, la cláusula bool `should` especificar una lista de consultas, cualquiera de las cuales debe ser cierta para que un documento se considere una coincidencia.

Este ejemplo compone dos consultas de coincidencia y devuelve todas las cuentas que no contienen "mill" ni "lane" en la `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "must_not": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

En el ejemplo anterior, la cláusula bool must_not especifica una lista de consultas, ninguna de las cuales debe ser cierta para que un documento se considere una coincidencia.

Podemos combinar las cláusulas must, should y must_not simultáneamente en una consulta bool. Además, podemos componer consultas bool dentro de cualquiera de estas cláusulas bool para imitar cualquier lógica booleana multinivel compleja.

Este ejemplo devuelve todas las cuentas que pertenecen a personas que tienen exactamente 40 años y no viven en Washington (`WA` para abreviar):

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "WA" } }
      ]
    }
  }
}'
```

Instalando Elasticsearch y Kibana en CentOS 7

Para ejecutar Elasticsearch, se requiere un Java Runtime Environment (JRE) en la máquina. Elasticsearch requiere Java 7 o superior y recomienda `Oracle JDK version 1.8.0_73` .

Por lo tanto, asegúrese de tener Java en su sistema. Si no, siga el procedimiento:

```
# Install wget with yum
yum -y install wget

# Download the rpm jre-8u60-linux-x64.rpm for 64 bit
wget --no-cookies --no-check-certificate --header "Cookie:
gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie"
"http://download.oracle.com/otn-pub/java/jdk/8u60-b27/jre-8u60-linux-x64.rpm"

# Download the rpm jre-8u101-linux-i586.rpm for 32 bit
wget --no-cookies --no-check-certificate --header "Cookie:
gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie"
"http://download.oracle.com/otn-pub/java/jdk/8u101-b13/jre-8u101-linux-i586.rpm"

# Install jre-.*.rpm
rpm -ivh jre-.*.rpm
```

Java ya debería estar instalado en su sistema centOS. Puedes comprobarlo con:

```
java -version
```

Descargar e instalar elasticsearch

```
# Download elasticsearch-2.3.5.rpm
wget
https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/rpm/elasticsearch/2.3.
2.3.5.rpm

# Install elasticsearch-.*.rpm
rpm -ivh elasticsearch-.*.rpm
```

Ejecutando elasticsearch como un servicio sistemático en el inicio

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch

# check the current status to ensure everything is okay.
systemctl status elasticsearch
```

Instalando Kibana

Primero importar GPG-key en rpm

```
sudo rpm --import http://packages.elastic.co/GPG-KEY-elasticsearch
```

Luego crea un repositorio local `kibana.repo`

```
sudo vi /etc/yum.repos.d/kibana.repo
```

Y añade el siguiente contenido:

```
[kibana-4.4]
name=Kibana repository for 4.4.x packages
baseurl=http://packages.elastic.co/kibana/4.4/centos
gpgcheck=1
gpgkey=http://packages.elastic.co/GPG-KEY-elasticsearch
enabled=1
```

Ahora instala la kibana siguiendo el comando:

```
yum -y install kibana
```

Comience con:

```
systemctl start kibana
```

Verificar el estado con:

```
systemctl status kibana
```

Puede ejecutarlo como un servicio de inicio.

```
systemctl enable kibana
```

Lea Empezando con Elasticsearch en línea:
https://riptutorial.com/es/elasticsearch/topic/941/empezando-con-elasticsearch

Capítulo 2: Agregaciones

Sintaxis

- "agregaciones": {- "<nombre_aggregación>": {- "<tipo de agregación>": {- <trabajo_registro> -} - [, "meta": {[<meta_data_body>]}]? - [, "agregaciones": {[<sub_aggregation>] +}]? -} - [, "<aggregation_name_2>": {...}] * -}

Examples

Agregación promedio

Esta es una agregación de métricas de un solo valor que calcula el promedio de los valores numéricos que se extraen de los documentos agregados.

```
POST /index/_search?
{
    "aggs" : {
        "avd_value" : { "avg" : { "field" : "name_of_field" } }
    }
}
```

La agregación anterior calcula la calificación promedio de todos los documentos. El tipo de agregación es promedio y la configuración del campo define el campo numérico de los documentos en los que se calculará el promedio. Lo anterior devolverá lo siguiente:

```
{
    ...
    "aggregations": {
        "avg_value": {
            "value": 75.0
        }
    }
}
```

El nombre de la agregación (avg_grade anterior) también sirve como la clave por la cual el resultado de la agregación se puede recuperar de la respuesta devuelta.

Agregación de cardinalidad

Una agregación de métricas de valor único que calcula un recuento aproximado de valores distintos. Los valores se pueden extraer de campos específicos en el documento o generados por un script.

```
POST /index/_search?size=0
{
    "aggs" : {
        "type_count" : {
```

```
            "cardinality" : {
                "field" : "type"
            }
        }
    }
}
```

Respuesta:

```
{
    ...
    "aggregations" : {
        "type_count" : {
            "value" : 3
        }
    }
}
```

Agregación de estadísticas extendidas

Una agregación de métricas de múltiples valores que calcula estadísticas sobre valores numéricos extraídos de los documentos agregados. Estos valores se pueden extraer de campos numéricos específicos en los documentos o generados por un script proporcionado.

Las agregaciones extended_stats son una versión extendida de la agregación de estadísticas, donde se agregan métricas adicionales, como sum_of_squares, variance, std_deviation y std_deviation_bounds.

```
{
    "aggs" : {
        "stats_values" : { "extended_stats" : { "field" : "field_name" } }
    }
}
```

Salida de muestra:

```
{
    ...

    "aggregations": {
        "stats_values": {
           "count": 9,
           "min": 72,
           "max": 99,
           "avg": 86,
           "sum": 774,
           "sum_of_squares": 67028,
           "variance": 51.55555555555556,
           "std_deviation": 7.180219742846005,
           "std_deviation_bounds": {
            "upper": 100.36043948569201,
            "lower": 71.63956051430799
           }
        }
    }
```

```
}
```

Lea Agregaciones en línea: https://riptutorial.com/es/elasticsearch/topic/10745/agregaciones

Capítulo 3: Analizadores

Observaciones

Los analizadores toman el texto de un campo de cadena y generan tokens que se utilizarán al realizar consultas.

Un analizador opera en una secuencia:

- `CharFilters` (cero o más)
- `Tokenizer` (uno)
- `TokenFilters` (cero o más)

El analizador se puede aplicar a las asignaciones de modo que cuando los campos se indexan, se realiza por token en lugar de en la cadena en su conjunto. Al realizar consultas, la cadena de entrada también se ejecutará a través del Analizador. Por lo tanto, si normaliza el texto en el Analizador, siempre coincidirá incluso si la consulta contiene una cadena no normalizada.

Examples

Cartografía

Se puede aplicar un Analizador a un mapeo usando "analizador", de manera predeterminada se usa el Analizador "estándar". Alternativamente, si no desea que se use ningún analizador (porque la tokenización o la normalización no serían útiles) puede especificar "índice": "not_analyzed"

```
PUT my_index
{
  "mappings": {
    "user": {
      "properties": {
        "name": {
          "type": "string"
          "analyzer":  "my_user_name_analyzer"
        },
        "id": {
          "type": "string",
          "index": "not_analyzed"
        }
      }
    }
  }
}
```

Campos múltiples

A veces puede ser útil tener múltiples índices distintos de un campo con diferentes analizadores. Puede utilizar la capacidad de campos múltiples para hacerlo.

```
PUT my_index
{
  "mappings": {
    "user": {
      "properties": {
        "name": {
          "type": "string"
          "analyzer": "standard",
          "fields": {
            "special": {
               "type": "string",
               "analyzer": "my_user_name_analyzer"
            },
            "unanalyzed": {
              "type": "string",
              "index": "not_analyzed"
            }
          }
        }
      }
    }
  }
}
```

Al realizar consultas, en lugar de simplemente usar "user.name" (que en este caso aún usaría el analizador estándar), puede usar "user.name.special" o "user.name.unanalyzed". Tenga en cuenta que el documento permanecerá sin cambios, esto solo afecta la indexación.

Analizadores

El análisis en elasticsearch entra en contexto cuando está dispuesto a analizar los datos en su índice.

Los analizadores nos permiten realizar lo siguiente:

- Abreviaturas
- Tallo
- Manejo tipográfico

Estaremos mirando a cada uno de ellos ahora.

1. **Abreviaturas** :

   Usando analizadores, podemos decirle a elasticsearch cómo tratar abreviaturas en nuestros datos, es decir, dr => Doctor, por lo que cada vez que busquemos una palabra clave de doctor en nuestro índice, elasticsearch también devolverá los resultados que dr ha mencionado en ellos.

2. **Tallo** :

   El uso de la derivación en analizadores nos permite usar palabras base para verbos modificados como

Palabra	Modificaciones
exigir	requisito requerido

3. **Manejo de errores tipográficos** :

   Los analizadores también brindan manejo de errores tipográficos mientras preguntan si estamos buscando una palabra en particular, digamos 'resurrección', luego elasticsearch devolverá los resultados en los cuales están presentes los errores tipográficos.

Palabra	Modificaciones
Resurrección	resurección, resurección

Analizadores en Elasticsearch

1. Estándar
2. Sencillo
3. Espacio en blanco
4. Detener
5. Palabra clave
6. Modelo
7. Idioma
8. Bola de nieve

Ignorar el analizador de casos

A veces, es posible que tengamos que ignorar el caso de nuestra consulta, con respecto a la coincidencia en el documento. En este caso, se puede usar un analizador para ignorar el caso durante la búsqueda. Cada campo deberá contener este analizador en su propiedad para que funcione:

```
"settings": {
        "analysis": {
            "analyzer": {
                "case_insensitive": {
               "tokenizer": "keyword",
               "filter": ["lowercase"]
            }
            }
        }
    }
```

Lea Analizadores en línea: https://riptutorial.com/es/elasticsearch/topic/6232/analizadores

Capítulo 4: API de búsqueda

Introducción

La API de búsqueda le permite ejecutar una consulta de búsqueda y recuperar resultados de búsqueda que coincidan con la consulta. La consulta puede proporcionarse utilizando una cadena de consulta simple como parámetro o utilizando un cuerpo de solicitud.

Examples

Enrutamiento

Al ejecutar una búsqueda, se transmitirá a todos los fragmentos de índice / índices (round robin entre réplicas). Los fragmentos en los que se buscará pueden controlarse proporcionando el parámetro de enrutamiento. Por ejemplo, al indexar tweets, el valor de enrutamiento puede ser el nombre de usuario:

```
curl -XPOST 'localhost:9200/twitter/tweet?routing=kimchy&pretty' -d'
{
    "user" : "kimchy",
    "postDate" : "2009-11-15T14:12:12",
    "message" : "trying out Elasticsearch"
}'
```

Buscar usando el cuerpo de solicitud

Las búsquedas también se pueden hacer en elasticsearch usando un DSL de búsqueda. El elemento de consulta dentro del cuerpo de la solicitud de búsqueda permite definir una consulta usando el DSL de consulta.

```
GET /my_index/type/_search
{
    "query" : {
        "term" : { "field_to_search" : "search_item" }
    }
}
```

Búsqueda múltiple

La opción multi_search nos permite buscar una consulta en varios campos a la vez.

```
GET /_search
{
  "query": {
    "multi_match" : {
      "query":    "text to search",
      "fields": [ "field_1", "field_2" ]
    }
```

```
  }
}
```

También podemos aumentar la puntuación de ciertos campos utilizando el operador de aumento (^) y usar comodines en el nombre del campo (*)

```
GET /_search
    {
      "query": {
        "multi_match" : {
          "query":    "text to search",
          "fields": [ "field_1^2", "field_2*" ]
        }
      }
    }
```

Búsqueda de URI, y resaltado

Una solicitud de búsqueda puede ejecutarse simplemente utilizando un URI proporcionando parámetros de solicitud. No todas las opciones de búsqueda están expuestas al ejecutar una búsqueda en este modo, pero puede ser útil para "pruebas de curvatura" rápidas.

```
GET Index/type/_search?q=field:value
```

Otra característica útil proporcionada es resaltar los éxitos de coincidencia en los documentos.

```
GET /_search
{
    "query" : {
        "match": { "field": "value" }
    },
    "highlight" : {
        "fields" : {
            "content" : {}
        }
    }
}
```

En el caso anterior, el campo particular se resaltará para cada golpe de búsqueda

Lea API de búsqueda en línea: https://riptutorial.com/es/elasticsearch/topic/8625/api-de-busqueda

Capítulo 5: Aprendiendo Elasticsearch con kibana

Introducción

Kibana es la herramienta de visualización de datos de front-end para elasticsearch. para instalar kibana consulte la documentación de kibana. Para ejecutar kibana en localhost, vaya a https: // localhost: 5601 y vaya a la consola de kibana.

Examples

Explora tu Clúster utilizando Kibana

La sintaxis del comando será del siguiente tipo:

```
<REST Verb> /<Index>/<Type>/<ID>
```

Ejecute el siguiente comando para explorar el clúster elasticsearch a través de la Consola Kibana.

- Para comprobar el estado del clúster.

```
GET /_cat/health?v
```

- Para listar todos los índices.

```
GET /_cat/indices?v
```

- Para crear un índice con nombre car

```
 PUT /car?pretty
```

- Para indexar el documento con nombre car de tipo externo usando id 1

```
PUT /car/external/1?pretty
{
  "name": "Tata Nexon"
}
```

La respuesta de la consulta anterior será:

```
{
  "_index": "car",
  "_type": "external",
  "_id": "1",
  "_version": 1,
```

```
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "created": true
}
```

- la recuperación del documento anterior se puede hacer usando:

```
GET /car/external/1?pretty
```

- Para borrar un índice

```
DELETE /car?pretty
```

Modifique sus datos de elasticsearch

Elasticsearch proporciona capacidades de manipulación y búsqueda de datos en casi tiempo real. En este ejemplo, tenemos operaciones de actualización, eliminación y procesamiento por lotes.

- Actualizando el mismo documento. Supongamos que ya hemos indexado un documento en / car / external / 1. Luego, al ejecutar el comando para indexar los datos se reemplaza el documento anterior.

```
PUT /car/external/1?pretty
{
  "name": "Tata Nexa"
}
```

El documento anterior del automóvil en la identificación 1 con el nombre "Tata Nexon" se actualizará con el nuevo nombre "Tata Nexa"

- indexando los datos con identificación explícita

```
POST /car/external?pretty
{
  "name": "Jane Doe"
}
```

  para indexar el documento sin un ID usamos el verbo **POST en** lugar del verbo **PUT** . Si no proporcionamos una identificación, elasticsearch generará una identificación aleatoria y luego la usará para indexar el documento.

- Actualización del documento anterior en un ID parcialmente.

```
POST /car/external/1/_update?pretty
{
  "doc": { "name": "Tata Nex" }
}
```

- Actualización del documento con información adicional.

```
POST /car/external/1/_update?pretty
{
  "doc": { "name": "Tata Nexon", "price": 1000000 }
}
```

- Actualización del documento mediante scripts simples.

```
POST /car/external/1/_update?pretty
{
  "script" : "ctx._source.price += 50000"
}
```

ctx._source se refiere al documento fuente actual que está a punto de actualizarse. La secuencia de comandos anterior solo proporciona una secuencia de comandos para actualizar al mismo tiempo.

- Borrando el documento

```
DELETE /car/external/1?pretty
```

Nota: eliminar un índice completo es más eficaz que eliminar todos los documentos mediante la función Eliminar por API de consulta

Procesamiento por lotes

Además de indexar la actualización y la eliminación del documento, elasticsearch también brinda la posibilidad de realizar cualquiera de las operaciones anteriores en lotes utilizando la API _**bulk** .

- para actualizar múltiples documentos usando _**bulk** API

```
POST /car/external/_bulk?pretty
{"index":{"_id":"1"}}
{"name": "Tata Nexon" }
{"index":{"_id":"2"}}
{"name": "Tata Nano" }
```

- para actualizar y borrar los documentos usando _**bulk** API

```
POST /car/external/_bulk?pretty
{"update":{"_id":"1"}}
{"doc": { "name": "Tata Nano" } }
{"delete":{"_id":"2"}}
```

Si una operación falla, la API masiva no se detiene. Ejecuta todas las operaciones y finalmente devuelve el informe para todas las operaciones.

Lea Aprendiendo Elasticsearch con kibana en línea:
https://riptutorial.com/es/elasticsearch/topic/10058/aprendiendo-elasticsearch-con-kibana

Capítulo 6: Comandos de Curl

Sintaxis

- curl -X <VERB> '<PROTOCOL>: // <HOST>: <PORT> / <PATH>? <QUERY_STRING>' -d '<BODY>'
- Dónde:
- VERBO: El método o verbo HTTP apropiado: GET, POST, PUT, HEAD o DELETE
- PROTOCOLO: http o https (si tiene un proxy https frente a Elasticsearch).
- HOST: El nombre de host de cualquier nodo en su grupo de Elasticsearch, o localhost para un nodo en su máquina local.
- PUERTO: el puerto que ejecuta el servicio HTTP Elasticsearch, que por defecto es 9200.
- PATH: API Endpoint (por ejemplo, _count devolverá el número de documentos en el clúster). La ruta puede contener múltiples componentes, como _cluster / stats o _nodes / stats / jvm
- QUERY_STRING: cualquier parámetro opcional de cadena de consulta (por ejemplo? Pretty imprimirá la respuesta JSON para que sea más fácil de leer).
- CUERPO: Un cuerpo de solicitud codificado en JSON (si la solicitud lo necesita).
- Referencia: Hablando con Elasticsearch: Elasticsearch Docs

Examples

Comando Curl para contar el número de documentos en el cluster

```
curl -XGET 'http://www.example.com:9200/myIndexName/_count?pretty'
```

Salida:

```
{
  "count" : 90,
  "_shards" : {
    "total" : 6,
    "successful" : 6,
    "failed" : 0
  }
}
```

El índice tiene 90 documentos dentro de él.

Enlace de referencia: Aquí

Recuperar un documento por ID

```
curl -XGET 'http://www.example.com:9200/myIndexName/myTypeName/1'
```

Salida:

```
{
    "_index" : "myIndexName",
    "_type" : "myTypeName",
    "_id" : "1",
    "_version" : 1,
    "found": true,
    "_source" : {
        "user" : "mrunal",
        "postDate" : "2016-07-25T15:48:12",
        "message" : "This is test document!"
    }
}
```

Enlace de referencia: Aquí

Crear un índice

```
curl -XPUT 'www.example.com:9200/myIndexName?pretty'
```

Salida:

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

Enlace de referencia: Aquí

Listar todos los índices

```
curl 'www.example.com:9200/_cat/indices?v'
```

salida:

```
health status index               pri rep docs.count docs.deleted store.size pri.store.size
green  open   logstash-2016.07.21   5   1       4760            0      4.8mb          2.4mb
green  open   logstash-2016.07.20   5   1       7232            0      7.5mb          3.7mb
green  open   logstash-2016.07.22   5   1      93528            0    103.6mb           52mb
green  open   logstash-2016.07.25   5   1      20683            0     41.5mb         21.1mb
```

Enlace de referencia: Aquí

Eliminar un índice

```
curl -XDELETE 'http://www.example.com:9200/myIndexName?pretty'
```

salida:

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

Enlace de referencia: Aquí

Listar todos los documentos en un índice

```
curl -XGET http://www.example.com:9200/myIndexName/_search?pretty=true&q=*:*
```

Esto utiliza la API de `Search` y devolverá todas las entradas bajo el índice `myIndexName` .

Enlace de referencia: Aquí

Lea Comandos de Curl en línea: https://riptutorial.com/es/elasticsearch/topic/3703/comandos-de-curl

Capítulo 7: Configuración de Elasticsearch

Observaciones

Elasticsearch viene con un conjunto de valores predeterminados que proporcionan una buena experiencia de desarrollo fuera de la caja. La declaración implícita allí es que no es necesariamente excelente para la producción, que debe adaptarse a sus propias necesidades y, por lo tanto, no puede predecirse.

La configuración predeterminada facilita la descarga y la ejecución de varios nodos *en la misma máquina* sin ningún cambio de configuración.

¿Dónde están los ajustes?

Dentro de cada instalación de Elasticsearch hay una `config/elasticsearch.yml` . Ahí es donde viven las siguientes configuraciones :

- `cluster.name`
  - El nombre del clúster al que se une el nodo. Todos los nodos en el mismo clúster **deben** compartir el mismo nombre.
  - Actualmente por defecto a `elasticsearch` .
- `node.*`
  - `node.name`
    - Si no se proporciona, se generará un nombre aleatorio *cada vez que se inicie el nodo* . Esto puede ser divertido, pero no es bueno para entornos de producción.
    - Los nombres *no* tienen que ser únicos, sino que **deben** ser únicos.
  - `node.master`
    - Un entorno booleano. Cuando es `true` , significa que el nodo es un nodo maestro elegible y puede ser *el* nodo maestro elegido.
    - El valor predeterminado es `true` , lo que significa que cada nodo es un nodo maestro elegible.
  - `node.data`
    - Un entorno booleano. Cuando es `true` , significa que el nodo almacena datos y maneja la actividad de búsqueda.
    - Por defecto es `true` .
- `path.*`
  - `path.data`
    - La ubicación en la que se escriben los archivos para el nodo. *Todos los nodos usan este directorio* para almacenar metadatos, pero los nodos de datos también lo usarán para almacenar / indexar documentos.
    - El valor predeterminado es `./data` .
      - Esto significa que los `data` se crearán para usted como un directorio de pares para `config` *dentro* del directorio de Elasticsearch.
  - `path.logs`
    - La ubicación donde se escriben los archivos de registro.
    - El valor predeterminado es `./logs` .
- `network.*`

    - `network.host`
        - El valor predeterminado es `_local_` , que es efectivamente `localhost` .
            - ¡Esto significa que, de forma predeterminada, los nodos no se pueden comunicar desde fuera de la máquina actual!
    - `network.bind_host`
        - Potencialmente una matriz, esto le dice a Elasticsearch qué direcciones de la máquina actual también deben enlazar sockets.
            - Es esta lista la que permite que las máquinas desde fuera de la máquina (por ejemplo, otros nodos en el clúster) se comuniquen con este nodo.
        - El valor predeterminado es `network.host` .
    - `network.publish_host`
        - Un host singular que se utiliza para anunciar a otros nodos la mejor manera de comunicarse con este nodo.
            - Al suministrar una matriz a `network.bind_host` , este debe ser el *único* host que se pretende utilizar para la comunicación entre nodos.
        - Por defecto a la red.
- `discovery.zen.*`
    - `discovery.zen.minimum_master_nodes`
        - Define quórum para elección magistral. Esto **debe** establecerse usando esta ecuación: `(M / 2) + 1` donde `M` es el número de nodos maestros *elegibles* (nodos que usan `node.master: true` implícitamente o explícitamente).
        - El valor predeterminado es `1` , que solo es válido para un clúster de un solo nodo.
    - `discovery.zen.ping.unicast.hosts`
        - El mecanismo para unir este nodo al resto de un cluster.
        - Esto *debe* incluir los nodos maestros elegibles para que un nodo pueda encontrar el resto del clúster.
        - El valor que se debe utilizar aquí es el `network.publish_host` de esos otros nodos.
        - El valor predeterminado es `localhost` , lo que significa que solo busca en la máquina local un clúster para unirse.

¿Qué tipo de configuraciones existen?

Elasticsearch ofrece tres tipos diferentes de configuraciones:

- Configuraciones de todo el cluster
    - Estas son configuraciones que se aplican a todo en el clúster, como todos los nodos o todos los índices.
- Configuraciones de nodo
    - Estas son configuraciones que se aplican solo al nodo actual.
- Configuración del índice
    - Estas son configuraciones que se aplican solo al índice.

Dependiendo de la configuración, puede ser:

- Cambiado dinámicamente en tiempo de ejecución

- Cambiado después de un reinicio (cierre / apertura) del índice
    - Algunas configuraciones de índice no requieren que el índice se cierre y se vuelva a abrir, pero podría requerir que el índice se vuelva a fusionar de manera forzosa para que se aplique la configuración.
        - El nivel de compresión de un índice es un ejemplo de este tipo de configuración. Se puede cambiar dinámicamente, pero solo los nuevos *segmentos* aprovechan el cambio. Entonces, si un índice no cambia, entonces nunca se aprovecha del cambio a menos que obligue al índice a recrear sus segmentos.
- Cambiado después de un reinicio del nodo
- Cambiado después de un reinicio del clúster
- Nunca cambió

Siempre revise la documentación de su versión de Elasticsearch para saber qué puede o no puede hacer con una configuración.

¿Cómo puedo aplicar la configuración?

Puede establecer la configuración de varias maneras, algunas de las cuales no se sugieren:

- Argumentos de línea de comando

En Elasticsearch 1.xy 2.x, puede enviar la mayoría de las configuraciones como Propiedades del sistema Java con el prefijo `es.` :

```
$ bin/elasticsearch -Des.cluster.name=my_cluster -Des.node.name=`hostname`
```

En Elasticsearch 5.x, esto cambia para evitar el uso de las Propiedades del sistema Java, en lugar de usar un tipo de argumento personalizado con `-E` tomando el lugar de `-Des.` :

```
$ bin/elasticsearch -Ecluster.name=my_cluster -Enode.name=`hostname`
```

Este enfoque para aplicar la configuración funciona muy bien cuando se usan herramientas como Puppet, Chef o Ansible para iniciar y detener el clúster. Sin embargo, funciona muy mal al hacerlo manualmente.

- Configuraciones de YAML
    - Se muestra en ejemplos
- Ajustes dinámicos
    - Se muestra en ejemplos

El orden en que se aplican las configuraciones es el orden más dinámico:

1. Ajustes transitorios
2. Ajustes persistentes
3. Configuraciones de línea de comando
4. Configuraciones YAML (estáticas)

Si la configuración se establece dos veces, una vez en cualquiera de esos niveles, el nivel más

alto tendrá efecto.

Examples

Configuraciones de búsqueda elástica estática

Elasticsearch utiliza un archivo de configuración YAML (Yet Another Markup Language) que se puede encontrar dentro del directorio predeterminado de Elasticsearch (las instalaciones RPM y DEB cambian esta ubicación, entre otras cosas).

Puede establecer configuraciones básicas en `config/elasticsearch.yml` :

```
# Change the cluster name. All nodes in the same cluster must use the same name!
cluster.name: my_cluster_name

# Set the node's name using the hostname, which is an environment variable!
# This is a convenient way to uniquely set it per machine without having to make
#  a unique configuration file per node.
node.name: ${HOSTNAME}

# ALL nodes should set this setting, regardless of node type
path.data: /path/to/store/data

# This is a both a master and data node (defaults)
node.master: true
node.data: true

# This tells Elasticsearch to bind all sockets to only be available
#  at localhost (default)
network.host: _local_
```

Configuración del clúster dinámico persistente

Si necesita aplicar una configuración dinámicamente después de que el clúster ya se haya iniciado, y realmente se puede establecer dinámicamente, entonces puede establecerla utilizando la API de `_cluster/settings` .

Las configuraciones persistentes son uno de los dos tipos de configuraciones a nivel de clúster que se pueden aplicar. Una configuración persistente **sobrevivirá** un reinicio completo del clúster.

Nota: No todas las configuraciones se pueden aplicar dinámicamente. Por ejemplo, el nombre del clúster no puede ser renombrado dinámicamente. La mayoría de las configuraciones de nivel de nodo tampoco se pueden establecer dinámicamente (porque no se pueden orientar individualmente).

Esta **no** es la API que se utiliza para establecer la configuración de nivel de índice. Se puede decir que la configuración es una configuración de nivel de índice porque debería comenzar con el `index.` . Configuraciones cuyo nombre está en forma de `indices.` *son* configuraciones de todo el clúster porque se aplican a todos los índices.

```
POST /_cluster/settings
```

```
{
  "persistent": {
    "cluster.routing.allocation.enable": "none"
  }
}
```

Advertencia : en Elasticsearch 1.xy 2.x, no puede *anular la configuración de* una configuración persistente.

Afortunadamente, esto se ha mejorado en Elasticsearch 5.x y ahora puedes eliminar una configuración configurándola en `null` :

```
POST /_cluster/settings
{
  "persistent": {
    "cluster.routing.allocation.enable": null
  }
}
```

Una configuración no establecida volverá a su valor predeterminado, o cualquier valor definido en un nivel de prioridad más bajo (por ejemplo, la configuración de la línea de comando).

Configuraciones transitorias de cluster dinámico

Si necesita aplicar una configuración dinámicamente después de que el clúster ya se haya iniciado, y realmente se puede establecer dinámicamente, entonces puede establecerla utilizando la API de `_cluster/settings` .

Las configuraciones transitorias son uno de los dos tipos de configuraciones a nivel de clúster que se pueden aplicar. Una configuración transitoria **no** sobrevivirá al reinicio completo del clúster.

Nota: No todas las configuraciones se pueden aplicar dinámicamente. Por ejemplo, el nombre del clúster no puede ser renombrado dinámicamente. La mayoría de las configuraciones de nivel de nodo tampoco se pueden establecer dinámicamente (porque no se pueden orientar individualmente).

Esta **no** es la API que se utiliza para establecer la configuración de nivel de índice. Se puede decir que la configuración es una configuración de nivel de índice porque debería comenzar con el `index.` . Configuraciones cuyo nombre está en forma de `indices.` *son* configuraciones de todo el clúster porque se aplican a todos los índices.

```
POST /_cluster/settings
{
  "transient": {
    "cluster.routing.allocation.enable": "none"
  }
}
```

Advertencia : en Elasticsearch 1.xy 2.x, no puede desactivar una configuración transitoria sin un reinicio completo del clúster.

Afortunadamente, esto se ha mejorado en Elasticsearch 5.x y ahora puedes eliminar una configuración configurándola en nula:

```
POST /_cluster/settings
{
  "transient": {
    "cluster.routing.allocation.enable": null
  }
}
```

Una configuración no establecida volverá a su valor predeterminado, o cualquier valor definido en un nivel de prioridad más bajo (por ejemplo, configuraciones `persistent`).

Configuración del índice

Las configuraciones de índice son aquellas configuraciones que se aplican a un solo índice. Dichos ajustes comenzarán con el `index.` . La excepción a esta regla es `number_of_shards` y `number_of_replicas` , que también existen en la forma de `index.number_of_shards` y `index.number_of_replicas` .

Como su nombre indica, la configuración de nivel de índice se aplica a un solo índice. Algunas configuraciones deben aplicarse en el momento de la creación porque no pueden cambiarse dinámicamente, como la configuración `index.number_of_shards` , que controla el número de fragmentos primarios para el índice.

```
PUT /my_index
{
  "settings": {
    "index.number_of_shards": 1,
    "index.number_of_replicas": 1
  }
}
```

o, en un formato más conciso, puede combinar prefijos de clave en cada uno `.` :

```
PUT /my_index
{
  "settings": {
    "index": {
      "number_of_shards": 1,
      "number_of_replicas": 1
    }
  }
}
```

Los ejemplos anteriores crearán un índice con la configuración suministrada. Puede cambiar dinámicamente la configuración por índice utilizando el `_settings` final de `_settings` índice. Por ejemplo, aquí cambiamos dinámicamente la configuración de slowlog *solo* para el nivel de advertencia:

```
PUT /my_index/_settings
```

```
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

Advertencia : Elasticsearch 1.xy 2.x no validaron muy estrictamente los nombres de configuración de nivel de índice. Si tuvieras un error tipográfico, o simplemente hicieras un ajuste, entonces lo aceptaría a ciegas, pero de lo contrario lo ignoraría. Elasticsearch 5.x valida estrictamente los nombres de configuración y rechazará cualquier intento de aplicar configuraciones de índice con una configuración (es) desconocida (debido a un error tipográfico o un complemento faltante). Ambas declaraciones se aplican a los cambios dinámicos en la configuración del índice y en el momento de la creación.

Configuración del índice dinámico para múltiples índices al mismo tiempo

Puede aplicar el mismo cambio que se muestra en el ejemplo de `Index Settings` a *todos* los índices existentes con una solicitud, o incluso con un subconjunto de ellos:

```
PUT /*/_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

o

```
PUT /_all/_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

o

```
PUT /_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
```

```
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

Si prefiere hacerlo también de forma más selectiva, puede seleccionar múltiples sin suministrar todos:

```
PUT /logstash-*,my_other_index,some-other-*/_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

Lea Configuración de Elasticsearch en línea:
https://riptutorial.com/es/elasticsearch/topic/3411/configuracion-de-elasticsearch

Capítulo 8: Diferencia entre bases de datos relacionales y elasticsearch

Introducción

Esto es para los lectores que provienen de antecedentes relacionales y desean aprender la búsqueda de elastics. Este tema muestra los casos de uso para los cuales las bases de datos relacionales no son una opción adecuada.

Examples

Terminología diferencia

Base de datos relacional	Elasticsearch
Base de datos	Índice
Mesa	Tipo
Fila / Grabar	Documento
Nombre de columna	campo

Sobre la tabla, aproximadamente se dibuja una analogía entre los elementos básicos de la base de datos relacional y la búsqueda de elastics.

Preparar

Teniendo en cuenta la siguiente estructura en una base de datos relacional:

```
create databse test;

use test;

create table product;

create table product (name varchar, id int PRIMARY KEY);

insert into product  (id,name) VALUES (1,'Shirt');

insert into product  (id,name) VALUES (2,'Red Shirt');

select * from product;

name       | id
-----------+----
Shirt      |  1
Red Shirt  |  2
```

Elasticsearch Equivalente:

```
POST test/product
{
    "id" : 1,
    "name" : "Shirt"
}

POST test/product
{
    "id" : 2,
    "name" : "Red Shirt"
}

GET test/product/_search


"hits": [
     {                                          ==============
        "_index": "test",                       ===> index    |
        "_type": "product",                     ===>type      |
        "_id": "AVzglFomaus3G2tXc6sB",                        |
        "_score": 1,                                          |
        "_source": {                                          |===> document
           "id": 2,                            ===>field      |
           "name": "Red Shirt"                 ===>field      |
        }                                                     |
     },                                         ==============
     {
        "_index": "test",
        "_type": "product",
        "_id": "AVzglD12aus3G2tXc6sA",
        "_score": 1,
        "_source": {
           "id": 1,
           "name": "Shirt"
        }
     }
  ]
```

Casos de uso donde las bases de datos relacionales no son adecuadas

- La esencia de la búsqueda reside en su orden. Todos quieren que los resultados de la búsqueda se muestren de tal manera que los mejores resultados se muestren en la parte superior. La base de datos relacional no tiene tal capacidad. Elasticsearch, por otro lado, muestra resultados en base a la relevancia por defecto.

  Preparar

  Igual que el utilizado en el ejemplo anterior.

  Planteamiento del problema

  Supongamos que el usuario quiere buscar `shirts` pero está interesado en camisas de color `red`. En ese caso, los resultados que contengan palabras clave de `red` y `shirts` deben aparecer en la parte superior. Luego se mostrarán los resultados de otras camisetas.

Solución mediante consulta de base de datos relacional

```
select * from product where name like '%Red%' or name like '%Shirt%';
```

Salida

```
name       | id
-----------+----
Shirt      |  1
Red Shirt  |  2
```

Solución de Elasticsearch

```
POST test/product/_search
{
     "query": {
          "match": {
            "name": "Red Shirt"
          }
     }
}
```

Salida

```
"hits": [
  {
     "_index": "test",
     "_type": "product",
     "_id": "AVzglFomaus3G2tXc6sB",
     "_score": 1.2422675,              ===> Notice this
     "_source": {
        "id": 2,
        "name": "Red Shirt"
     }
  },
  {
     "_index": "test",
     "_type": "product",
     "_id": "AVzglD12aus3G2tXc6sA",
     "_score": 0.25427115,             ===> Notice this
     "_source": {
        "id": 1,
        "name": "Shirt"
     }
  }
 ]
```

Conclusión

Como podemos ver arriba, la base de datos relacional ha arrojado resultados en un orden aleatorio, mientras que Elasticsearch devuelve los resultados en orden decreciente de `_score` que se calcula sobre la base de la relevancia.

- Tendemos a cometer errores al introducir la cadena de búsqueda. Hay casos en que el

usuario ingresa un parámetro de búsqueda incorrecto. Las bases de datos relacionales no manejarán tales casos. Elasticsearch al rescate.

Preparar

Igual que el utilizado en el ejemplo anterior.

Planteamiento del problema

Supongamos que el usuario quiere buscar `shirts` pero él entra en una palabra incorrecta `shrt` por error. El usuario aún espera ver los resultados de la camiseta .

Solución mediante consulta de base de datos relacional

```
select * from product where name like '%shrt%';
```

Salida

```
No results found
```

Solución de Elasticsearch

```
POST /test/product/_search

 {
    "query": {
      "match": {
        "name": {
          "query": "shrt",
          "fuzziness": 2,
          "prefix_length": 0
         }
      }
    }
 }
```

Salida

```
 "hits": [
  {
     "_index": "test",
     "_type": "product",
     "_id": "AVzglD12aus3G2tXc6sA",
     "_score": 1,
     "_source": {
        "id": 1,
        "name": "Shirt"
     }
  },
  {
     "_index": "test",
     "_type": "product",
     "_id": "AVzglFomaus3G2tXc6sB",
     "_score": 0.8784157,
     "_source": {
```

```
        "id": 2,
        "name": "Red Shirt"
    }
  }
]
```

Conclusión

Como podemos ver arriba, la base de datos relacional no ha devuelto resultados por una palabra incorrecta buscada, mientras que Elasticsearch que usa su consulta `fuzzy` especial devuelve resultados.

Lea Diferencia entre bases de datos relacionales y elasticsearch en línea: https://riptutorial.com/es/elasticsearch/topic/10632/diferencia-entre-bases-de-datos-relacionales-y-elasticsearch

Capítulo 9: Diferencia entre índices y tipos

Observaciones

Es fácil ver el `type` s como una tabla en una base de datos SQL, donde el `index` es la base de datos SQL. Sin embargo, esa no es una buena manera de abordar el `type` s.

Todo sobre los tipos

De hecho, los tipos son, *literalmente,* solo un campo de metadatos agregado a cada documento por `_type` : `_type` . Los ejemplos anteriores crearon dos tipos: `my_type` y `my_other_type` . Eso significa que cada documento asociado con los tipos tiene un campo adicional automáticamente definido como `"_type": "my_type"` ; Esto se indexa con el documento, lo que lo convierte en un *campo de búsqueda o de filtro* , pero no afecta al documento sin formato, por lo que su aplicación no necesita preocuparse por ello.

Todos los tipos viven en el mismo índice y, por lo tanto, en los mismos fragmentos colectivos del índice. Incluso a nivel de disco, viven en los mismos archivos. La única separación que proporciona la creación de un segundo tipo es una lógica. Cada tipo, ya sea único o no, debe existir en las asignaciones y todas esas asignaciones deben existir en su estado de agrupación. Esto consume memoria y, si cada tipo se actualiza dinámicamente, consume el rendimiento a medida que cambian las asignaciones.

Como tal, es una buena práctica definir solo un tipo a menos que realmente necesite otros tipos. Es común ver escenarios donde son deseables múltiples tipos. Por ejemplo, imagina que tienes un índice de autos. Puede ser útil para usted descomponerlo en varios tipos:

- BMW
- caza
- honda
- mazda
- mercedes
- nissan
- guardabosques
- toyota
- ...

De esta manera puede buscar todos los autos o limitarlos por fabricante a pedido. La diferencia entre esas dos búsquedas es tan simple como:

```
GET /cars/_search
```

y

```
GET /cars/bmw/_search
```

Lo que no es obvio para los nuevos usuarios de Elasticsearch es que la segunda forma es una especialización de la primera forma. Literalmente se reescribe a:

```
GET /cars/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "term" : {
            "_type": "bmw"
          }
        }
      ]
    }
  }
}
```

Simplemente filtra cualquier documento que no haya sido indexado con un campo `_type` cuyo valor fue `bmw` . Dado que cada documento está indexado con su tipo como el campo `_type` , esto sirve como un filtro bastante simple. Si se proporcionó una búsqueda real en cualquiera de los dos ejemplos, entonces el filtro se agregará a la búsqueda completa según corresponda.

Como tal, si los tipos son idénticos, es mucho mejor suministrar un solo tipo (p. Ej., `manufacturer` en este ejemplo) e ignorarlo de manera efectiva. Luego, dentro de cada documento, proporcione explícitamente un campo llamado `make` *o el nombre que prefiera* y filtre manualmente cada vez que quiera limitarlo. Esto reducirá el tamaño de sus asignaciones a `1/n` donde `n` es el número de tipos separados. Añade otro campo a cada documento, en beneficio de un mapeo simplificado.

En Elasticsearch 1.xy 2.x, este campo debe definirse como

```
PUT /cars
{
  "manufacturer": { <1>
    "properties": {
      "make": { <2>
        "type": "string",
        "index": "not_analyzed"
      }
    }
  }
}
```

1. El nombre es arbitrario.
2. El nombre es arbitrario *y* podría coincidir con el nombre de tipo si lo deseara también.

En Elasticsearch 5.x, lo anterior seguirá funcionando (está en desuso), pero la mejor manera es usar:

```
PUT /cars
{
  "manufacturer": { <1>
    "properties": {
      "make": { <2>
```

```
        "type": "keyword"
      }
    }
  }
}
```

1. El nombre es arbitrario.
2. El nombre es arbitrario *y* podría coincidir con el nombre de tipo si lo deseara también.

Los tipos deben usarse con moderación dentro de sus índices, ya que aumentan las asignaciones de índice, generalmente sin mucho beneficio. Debe tener al menos uno, pero no hay nada que diga que debe tener más de uno.

Preguntas comunes

- ¿Qué sucede si tengo dos (o más) tipos que son en su mayoría idénticos, pero que tienen algunos campos únicos por tipo?

En el nivel de índice, no hay diferencia entre un tipo que se usa con unos pocos campos que se utilizan poco *y* entre varios tipos que comparten un grupo de campos no dispersos con unos pocos no compartidos (lo que significa que el otro tipo ni siquiera usa el campo). (s)).

Dicho de otra manera: un campo poco utilizado es escaso en todo el índice, *independientemente de los tipos* . La escasez no beneficia, o realmente perjudica, al índice solo porque está definido en un tipo separado.

Simplemente debe combinar estos tipos y agregar un campo de tipo separado.

- ¿Por qué tipos separados necesitan definir campos exactamente de la misma manera?

Debido a que cada campo realmente solo se define una vez en el nivel de Lucene, independientemente de cuántos tipos haya. El hecho de que existan tipos es una característica de Elasticsearch y es *solo* una separación lógica.

- ¿Puedo definir tipos separados con el mismo campo definido de manera diferente?

No. Si logra encontrar una manera de hacerlo en ES 2.x o posterior, debe abrir un informe de error . Como se señaló en la pregunta anterior, Lucene los ve a todos como un solo campo, por lo que no hay manera de hacer que esto funcione adecuadamente.

ES 1.x dejó esto como un requisito implícito, lo que permitió a los usuarios crear condiciones en las que las asignaciones de un fragmento en un índice realmente diferían de otro fragmento en el mismo índice. Esto fue efectivamente una condición de carrera y *podría* llevar a problemas inesperados.

Excepciones a la Regla

- Los documentos padre / hijo **requieren** tipos separados para usarse dentro del mismo índice.

    - El padre vive en un tipo.
    - El niño vive en un tipo separado (pero cada niño vive en el mismo *fragmento* que su padre).
- Los casos de uso de nichos extremos en los que la creación de toneladas de índices no es deseable y el impacto de campos dispersos es preferible a la alternativa.
    - Por ejemplo, el complemento de monitoreo de Elasticsearch, Marvel (1.xy 2.x) o X-Pack Monitoring (5.x +), monitorea a Elasticsearch en busca de cambios en el clúster, nodos, índices, índices específicos (el nivel del índice), e incluso los fragmentos. Podría crear más de 5 índices cada día para aislar los documentos que tienen asignaciones únicas *o* podría ir en contra de las mejores prácticas para reducir la carga de clústeres al compartir un índice (nota: la cantidad de asignaciones definidas es efectivamente la misma, pero la cantidad de índices creados). se reduce de `n` a 1).
    - Este es un escenario avanzado, pero debe tener en cuenta las definiciones de campos compartidos entre los tipos.

Examples

Creando explícitamente un índice con un tipo

Ejemplo utiliza HTTP básico, que se traduce fácilmente a cURL y otras aplicaciones HTTP. También coinciden con la sintaxis de Sense , que cambiará de nombre a Consola en Kibana 5.0.

Nota: El ejemplo inserta `<#>` para ayudar a llamar la atención sobre las partes. ¡Aquellos deben ser eliminados si lo copia!

```
PUT /my_index <1>
{
  "mappings": {
    "my_type": { <2>
      "properties": {
        "field1": {
          "type": "long"
        },
        "field2": {
          "type": "integer"
        },
        "object1": {
          "type": "object",
          "properties": {
            "field1" : {
              "type": "float"
            }
          }
        }
      }
    }
  },
  "my_other_type": {
    "properties": {
      "field1": {
        "type": "long" <3>
      },
      "field3": { <4>
```

```
        "type": "double"
      }
    }
  }
}
```

1. Esto es crear el `index` utilizando el punto final crear índice.
2. Esto está creando el `type` .
3. ¡Los campos compartidos en el `type` s dentro del mismo `index` **deben** compartir la misma definición! ES 1.x no aplicó estrictamente este comportamiento, pero era un requisito implícito. ES 2.x y superior imponen estrictamente este comportamiento.
4. Los campos únicos en el `type` s están bien.

Los índices (o índices) *contienen* tipos. Los tipos son un mecanismo conveniente para separar documentos, pero requieren que defina, ya sea de forma dinámica / automática o explícita, una asignación para cada tipo que use. Si define 15 tipos en un índice, entonces tiene 15 asignaciones únicas.

Vea las observaciones para obtener más detalles sobre este concepto y por qué puede o no desear usar tipos.

Crear dinámicamente un índice con un tipo

Ejemplo utiliza HTTP básico, que se traduce fácilmente a cURL y otras aplicaciones HTTP. También coinciden con la sintaxis de Sense , que cambiará de nombre a Consola en Kibana 5.0.

Nota: El ejemplo inserta `<#>` para ayudar a llamar la atención sobre las partes. ¡Aquellos deben ser eliminados si lo copia!

```
DELETE /my_index <1>

PUT /my_index/my_type/abc123 <2>
{
  "field1" : 1234, <3>
  "field2" : 456,
  "object1" : {
    "field1" : 7.8 <4>
  }
}
```

1. En caso de que ya exista (debido a un ejemplo anterior), elimine el índice.
2. `my_index` un documento en el índice, `my_index` , con el tipo, `my_type` y el ID `abc123` (podría ser numérico, pero siempre es una cadena).
    - De forma predeterminada, la creación de índices dinámicos se habilita simplemente indexando un documento. Esto es ideal para entornos de desarrollo, pero no es necesariamente bueno para entornos de producción.
3. Este campo es un número entero, por lo que la primera vez que se ve, debe asignarse. Elasticsearch siempre asume el tipo *más ancho* para cualquier tipo entrante, por lo que este se asignaría como un `long` lugar de un `integer` o un `short` (ambos podrían contener `1234` y `456`).

4. Lo mismo es cierto para este campo también. Se asignará como un `double` lugar de un `float` como desee.

Este índice y tipo creados dinámicamente coinciden aproximadamente con la asignación definida en el primer ejemplo. Sin embargo, es fundamental entender cómo `<3>` y `<4>` afectan las asignaciones definidas automáticamente.

Puedes seguir esto agregando otro tipo dinámicamente al mismo índice:

```
PUT /my_index/my_other_type/abc123 <1>
{
  "field1": 91, <2>
  "field3": 4.567
}
```

1. El tipo es la única diferencia con respecto al documento anterior. La identificación es la misma y eso está bien! No tiene ninguna relación con los otros `abc123` aparte de eso, *pasa* a ser en el mismo índice.
2. `field1` ya existe en el índice, por lo que *debe* ser el mismo tipo de campo definido en los otros tipos. El envío de un valor que era una cadena o que no es un entero fallaría (por ejemplo, `"field1": "this is some text"` o `"field1": 123.0`).

Esto crearía dinámicamente las asignaciones para `my_other_type` dentro del mismo índice, `my_index` .

Nota: *Siempre* es más rápido definir asignaciones por adelantado en lugar de que Elasticsearch lo realice dinámicamente en el momento del índice.

El resultado final de la indexación de ambos documentos sería similar al primer ejemplo, pero los tipos de campo serían diferentes y, por lo tanto, un poco inútil:

```
GET /my_index/_mappings <1>
{
  "mappings": {
    "my_type": { <2>
      "properties": {
        "field1": {
          "type": "long"
        },
        "field2": {
          "type": "long" <3>
        },
        "object1": {
          "type": "object",
          "properties": {
            "field1" : {
              "type": "double" <4>
            }
          }
        }
      }
    }
  },
  "my_other_type": { <5>
```

```
    "properties": {
      "field1": {
        "type": "long"
      },
      "field3": {
        "type": "double"
      }
    }
  }
}
```

1. Esto utiliza el `_mappings` final `_mappings` para obtener las asignaciones del índice que creamos.
2. Creamos dinámicamente `my_type` en el primer paso de este ejemplo.
3. `field2` ahora es un `long` lugar de un `integer` porque no lo definimos por adelantado. Esto *puede* resultar un desperdicio en el almacenamiento en disco.
4. `object1.field1` ahora es `double` por la misma razón que # 3 con las mismas ramificaciones que # 3.
    - Técnicamente, un `long` puede ser comprimido en muchos casos. Sin embargo, un `double` no se puede comprimir debido a que es un número de punto flotante.
5. También creamos dinámicamente `my_other_type` en el segundo paso de este ejemplo. Su mapeo pasa a ser el mismo porque ya usábamos `long` y `double` .
    - Recuerde que `field1` *debe* coincidir con la definición de `my_type` (y lo hace).
    - `field3` es exclusivo de este tipo, por lo que no tiene tal restricción.

Lea Diferencia entre índices y tipos en línea:
https://riptutorial.com/es/elasticsearch/topic/3412/diferencia-entre-indices-y-tipos

Capítulo 10: Interfaz Python

Parámetros

Parámetro	Detalles
Hospedadores	Array de hosts en forma de objeto que contiene claves `host` y `port` . El `host` predeterminado es 'localhost' y el `port` es 9200. Una entrada de muestra se parece a `[{"host": "ip of es server", "port": 9200}]`
sniff_on_start	Booleano si desea que el cliente detecte nodos en el inicio, rastrear significa obtener una lista de nodos en el clúster elasticsearch
sniff_on_connection_fail	Booleano para activar el rastreo si la conexión falla cuando el cliente está activo
sniffer_timeout	diferencia de tiempo en segundos entre cada aspiración
sniff_timeout	tiempo para una sola solicitud de sniffing en segundos
retry_on_timeout	Booelan para si el cliente debe tener un tiempo de espera en contacto con un nodo diferente de elasticsearch o simplemente lanzar un error
http_auth	La autenticación http básica se puede proporcionar aquí en forma de `username:password` de `username:password`

Examples

Indexando un documento (es decir, añadiendo una muestra)

Instale la biblioteca de Python necesaria a través de:

```
$ pip install elasticsearch
```

Conéctese a Elasticsearch, cree un documento (por ejemplo, entrada de datos) e "indexe" el documento utilizando Elasticsearch.

```
from datetime import datetime
from elasticsearch import Elasticsearch

# Connect to Elasticsearch using default options (localhost:9200)
es = Elasticsearch()
```

```
# Define a simple Dictionary object that we'll index to make a document in ES
doc = {
    'author': 'kimchy',
    'text': 'Elasticsearch: cool. bonsai cool.',
    'timestamp': datetime.now(),
}

# Write a document
res = es.index(index="test-index", doc_type='tweet', id=1, body=doc)
print(res['created'])

# Fetch the document
res = es.get(index="test-index", doc_type='tweet', id=1)
print(res['_source'])

# Refresh the specified index (or indices) to guarantee that the document
#  is searchable (avoid race conditions with near realtime search)
es.indices.refresh(index="test-index")

# Search for the document
res = es.search(index="test-index", body={"query": {"match_all": {}}})
print("Got %d Hits:" % res['hits']['total'])

# Show each "hit" or search response (max of 10 by default)
for hit in res['hits']['hits']:
    print("%(timestamp)s %(author)s: %(text)s" % hit["_source"])
```

Conexión a un cluster

```
es = Elasticsearch(hosts=hosts, sniff_on_start=True, sniff_on_connection_fail=True,
sniffer_timeout=60, sniff_timeout=10, retry_on_timeout=True)
```

Creando un índice vacío y configurando el mapeo

En este ejemplo, creamos un índice vacío (no indexamos ningún documento en él) definiendo su mapeo.

Primero, creamos una instancia de `ElasticSearch` y luego definimos el mapeo de nuestra elección. A continuación, verificamos si el índice existe y, de no ser así, lo creamos especificando los parámetros de `index` y `body` que contienen el nombre del índice y el cuerpo de la asignación, respectivamente.

```
from elasticsearch import Elasticsearch

# create an ElasticSearch instance
es = Elasticsearch()
# name the index
index_name = "my_index"
# define the mapping
mapping = {
    "mappings": {
        "my_type": {
                "properties": {
                    "foo": {'type': 'text'},
                    "bar": {'type': 'keyword'}
```

```
                }
            }
        }
    }

# create an empty index with the defined mapping - no documents added
if not es.indices.exists(index_name):
    res = es.indices.create(
        index=index_name,
        body=mapping
    )
    # check the response of the request
    print(res)
    # check the result of the mapping on the index
    print(es.indices.get_mapping(index_name))
```

Actualización parcial y actualización por consulta

Actualización parcial: se utiliza cuando se necesita una actualización parcial del documento, es decir, en el siguiente ejemplo, el `name` de campo del documento con id `doc_id` se actualizará a "John". Tenga en cuenta que si falta el campo, solo se agregará al documento.

```
doc = {
    "doc": {
        "name": "John"
    }
}
es.update(index='index_name',
          doc_type='doc_name',
          id='doc_id',
          body=doc)
```

Actualizar por consulta: se utiliza cuando es necesario para actualizar documentos que satisfacen una condición, es decir, en el siguiente ejemplo, actualizamos la antigüedad de los documentos cuyo campo de `name` coincide con 'John'.

```
q = {
  "script": {
    "inline": "ctx._source.age=23",
    "lang": "painless"
  },
  "query": {
    "match": {
        "name": "John"
    }
  }
}

es.update_by_query(body=q,
                   doc_type='doc_name',
                   index='index_name')
```

Lea Interfaz Python en línea: https://riptutorial.com/es/elasticsearch/topic/2068/interfaz-python

Capítulo 11: Racimo

Observaciones

El estado del clúster proporciona mucha información sobre el clúster, como la cantidad de fragmentos asignados ("activos") y cuántos están sin asignar y reubicados. Además, proporciona el número actual de nodos y nodos de datos en el clúster, lo que puede permitirle sondear los nodos faltantes (por ejemplo, si espera que sean `15` , pero solo muestra `14` , entonces falta un nodo) .

Para alguien que sepa sobre Elasticsearch, los fragmentos "asignados" y "no asignados" pueden ayudarlos a localizar problemas.

El campo más común verificado desde Cluster Health es el `status` , que puede estar en uno de tres estados:

- rojo
- amarillo
- verde

Cada uno de los colores significa uno, y solo uno, algo muy simple:

1. Rojo indica que falta *al menos* un fragmento primario.
    - Un fragmento primario faltante significa que no se puede usar un índice para escribir (indexar) datos nuevos en la mayoría de los casos.
        - Técnicamente, aún puede indexar a cualquier fragmento primario que esté disponible en ese índice, pero prácticamente significa que no puede hacerlo porque generalmente no controla qué fragmento recibe un documento determinado.
        - La búsqueda aún es posible contra un clúster rojo, pero significa que obtendrá resultados parciales si faltan fragmentos en algún índice que busque.
    - En circunstancias normales, solo significa que se está asignando el fragmento primario (`initializing_shards`).
    - Si un nodo acaba de abandonar el clúster (por ejemplo, porque la máquina que lo ejecuta perdió energía), tiene sentido que le falten algunos fragmentos primarios *temporalmente* .
        - Cualquier fragmento de réplica para ese fragmento primario se promocionará para ser el fragmento primario en este escenario.
2. El amarillo indica que todos los fragmentos primarios están activos, pero falta *al menos* un fragmento de réplica.
    - Una réplica que falta solo afecta a la indexación si la configuración de coherencia requiere que afecte a la indexación.
        - De forma predeterminada, solo hay una réplica para cualquier primario y la indexación puede ocurrir con una sola réplica faltante.
    - En circunstancias normales, solo significa que el fragmento de réplica se está asignando (`initializing_shards`).

- Un clúster de un nodo con réplicas habilitadas *siempre* será amarillo *en el mejor de los casos* . Puede ser rojo si aún no se ha asignado un fragmento primario.
    - Si solo tiene un solo nodo, tiene sentido deshabilitar las réplicas porque no espera ninguna. Entonces puede ser verde.

3. Verde indica que todos los fragmentos están activos.
    - La única actividad de fragmento permitida para un clúster verde es `relocating_shards` .
    - Los nuevos índices, y por lo tanto los nuevos fragmentos, harán que el grupo pase de rojo a amarillo a verde, ya que cada fragmento se asigna (primero el primario, lo hace amarillo y luego las réplicas si es posible, haciéndolo verde).
        - En Elasticsearch 5.x y versiones posteriores, los nuevos índices **no** harán que su clúster se vuelva rojo a menos que se demore demasiado en asignarlos.

Examples

Salud de grupo tabular legible por humanos con encabezados

Ejemplo utiliza la sintaxis básica de HTTP. Cualquier `<#>` en el ejemplo debe eliminarse al copiarlo.

Puede usar las API `_cat` para obtener un resultado tabular, legible por el ser humano, por varias razones.

```
GET /_cat/health?v <1>
```

1. El `?v` es opcional, pero implica que desea una salida "detallada".

`_cat/health` existe desde Elasticsearch 1.x, pero aquí hay un ejemplo de su salida de Elasticsearch 5.x:

Con salida verbosa:

```
epoch      timestamp cluster       status node.total node.data shards pri relo init unassign
pending_tasks max_task_wait_time active_shards_percent
1469302011 15:26:51  elasticsearch yellow          1         1     45  45    0    0       44
0                  -                  50.6%
```

Salud del grupo tabular legible por humanos sin encabezados

Ejemplo utiliza la sintaxis básica de HTTP. Cualquier `<#>` en el ejemplo debe eliminarse al copiarlo.

Puede usar las API `_cat` para obtener un resultado tabular, legible por el ser humano, por varias razones.

```
GET /_cat/health <1>
```

`_cat/health` existe desde Elasticsearch 1.x, pero aquí hay un ejemplo de su salida de

Elasticsearch 5.x:

Sin salida verbosa:

```
1469302245 15:30:45 elasticsearch yellow 1 1 45 45 0 0 44 0 - 50.6%
```

Salud de grupo tabular legible por humanos con encabezados seleccionados

Ejemplo utiliza la sintaxis básica de HTTP. Cualquier `<#>` en el ejemplo debe eliminarse al copiarlo.

Como la mayoría de `_cat` API de `_cat` en Elasticsearch, la API responde selectivamente con un conjunto predeterminado de campos. Sin embargo, existen otros campos de la API si los desea:

```
GET /_cat/health?help <1>
```

1. `?help` hace que la API devuelva los campos (y los nombres cortos), así como una breve descripción.

`_cat/health` existe desde Elasticsearch 1.x, pero aquí hay un ejemplo de su salida de Elasticsearch 5.x:

Campos disponibles a partir de la fecha de creación de este ejemplo:

```
epoch                | t,time                                | seconds since 1970-01-01
00:00:00
timestamp            | ts,hms,hhmmss                         | time in HH:MM:SS

cluster              | cl                                    | cluster name

status               | st                                    | health status

node.total           | nt,nodeTotal                          | total number of nodes

node.data            | nd,nodeData                           | number of nodes that can
store data
shards               | t,sh,shards.total,shardsTotal         | total number of shards

pri                  | p,shards.primary,shardsPrimary        | number of primary shards

relo                 | r,shards.relocating,shardsRelocating  | number of relocating nodes

init                 | i,shards.initializing,shardsInitializing | number of initializing
nodes
unassign             | u,shards.unassigned,shardsUnassigned  | number of unassigned shards

pending_tasks        | pt,pendingTasks                       | number of pending tasks

max_task_wait_time   | mtwt,maxTaskWaitTime                  | wait time of longest task
pending
active_shards_percent | asp,activeShardsPercent              | active number of shards in
percent
```

Luego puede usar esto para imprimir solo esos campos:

```
GET /_cat/health?h=timestamp,cl,status&v <1>
```

1. `h=...` define la lista de campos que desea que se devuelvan.
2. `v` (detallado) define que desea que imprima los encabezados.

La salida de una instancia de Elasticsearch 5.x:

```
timestamp cl            status
15:38:00  elasticsearch yellow
```

Salud de clúster basada en JSON

Ejemplo utiliza la sintaxis básica de HTTP. Cualquier `<#>` en el ejemplo debe eliminarse al copiarlo.

Las API `_cat` menudo son convenientes para que los humanos obtengan detalles de un vistazo sobre el clúster. Pero con frecuencia se desea que la salida se pueda analizar de forma consistente para usar con el software. En general, las API de JSON están diseñadas para este propósito.

```
GET /_cluster/health
```

`_cluster/health` ha existido desde Elasticsearch 1.x, pero aquí hay un ejemplo de su salida de Elasticsearch 5.x:

```
{
  "cluster_name": "elasticsearch",
  "status": "yellow",
  "timed_out": false,
  "number_of_nodes": 1,
  "number_of_data_nodes": 1,
  "active_primary_shards": 45,
  "active_shards": 45,
  "relocating_shards": 0,
  "initializing_shards": 0,
  "unassigned_shards": 44,
  "delayed_unassigned_shards": 0,
  "number_of_pending_tasks": 0,
  "number_of_in_flight_fetch": 0,
  "task_max_waiting_in_queue_millis": 0,
  "active_shards_percent_as_number": 50.56179775280899
}
```

Lea Racimo en línea: https://riptutorial.com/es/elasticsearch/topic/2069/racimo

Creditos

S. No	Capítulos	Contributors
1	Empezando con Elasticsearch	Ahsanul Haque, Berto, Community, DJanssens, Dulguun, igo, KartikKannapur, manishrw, mightyteja, noscreenname, Onur, rafa.ferreira, RustyBuckets, sarvajeetsuman, SeinopSys, Shivkumar Mallesappa, Stephan-v, Suhas K, Sumit Kumar, Trilarion
2	Agregaciones	Sid1199
3	Analizadores	Bhushan Gadekar, Sid1199, Thomas
4	API de búsqueda	aerokite, Sid1199
5	Aprendiendo Elasticsearch con kibana	sarvajeetsuman
6	Comandos de Curl	Fawix, Mrunal Pagnis, Mrunal Pagnis
7	Configuración de Elasticsearch	pickypg
8	Diferencia entre bases de datos relacionales y elasticsearch	Richa
9	Diferencia entre índices y tipos	pickypg
10	Interfaz Python	aidan.plenert.macdonald, christinabo, KartikKannapur, pickypg, Sumit Kumar
11	Racimo	Gerardo Rochín, pickypg