EBook Gratuito

APPRENDIMENTO
Elasticsearch

Free unaffiliated eBook created from
Stack Overflow contributors.

#elasticsear

ch

Sommario

Di

Capitolo 1: Iniziare con Elasticsearch

Osservazioni

Elasticsearch è un avanzato server di ricerca open source basato su Lucene e scritto in Java.

Fornisce funzionalità di ricerca completa e parziale distribuita, basata su query e basata sulla geolocalizzazione accessibile tramite un'API REST HTTP.

Versioni

Versione	Data di rilascio
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
1.2.0	2014/05/22

Versione	Data di rilascio
1.1.0	2014/03/25
1.0.0	2014/02/14

Examples

Installazione di Elasticsearch su Ubuntu 14.04

Prerequisiti

Per eseguire Elasticsearch, è richiesto un Java Runtime Environment (JRE) sulla macchina. Elasticsearch richiede Java 7 o versioni successive e raccomanda `Oracle JDK version 1.8.0_73` .

Installa 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
```

Controlla la versione di Java

```
java -version
```

Scarica e installa il pacchetto

Usando i binari

1. Scarica l'ultima versione stabile di elasticsearch qui .
2. Decomprimere il file ed eseguire

Linux:

```
$ bin/elasticsearch
```

Utilizzando apt-get

Un'alternativa al download di elasticsearch dal sito Web è l'installazione, utilizzando `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
```

Installazione di elasticsearch versione 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
```

Esecuzione come servizio su Linux:

Dopo aver installato quanto sopra non si avvia da solo. quindi abbiamo bisogno di avviarlo come servizio. Come avviare o interrompere Elasticsearch dipende dal fatto che il tuo sistema utilizzi SysV init o systemd. puoi controllarlo con il seguente comando.

```
ps -p 1
```

Se la tua distribuzione utilizza SysV init, dovrai eseguire:

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

Altrimenti se la tua distribuzione sta usando systemd:

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

Esegui il comando `CURL` dal tuo browser o da un client REST, per verificare se Elasticsearch è stato installato correttamente.

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

Installazione di Elasticsearch su Windows

Prerequisiti

La versione per Windows di Elasticsearch può essere ottenuta da questo link: https://www.elastic.co/downloads/elasticsearch . L'ultima versione stabile è sempre al top.

Poiché stiamo installando su Windows, abbiamo bisogno dell'archivio `.ZIP` . Fare clic sul collegamento nella sezione `Downloads:` e salvare il file sul computer.

Questa versione di elastico è "portatile", il che significa che non è necessario eseguire un programma di installazione per utilizzare il programma. Decomprimi il contenuto del file in una

posizione che puoi facilmente ricordare. Per la dimostrazione, supponiamo che tu abbia decompresso tutto in `C:\elasticsearch` .

Si noti che l'archivio contiene una cartella denominata `elasticsearch-<version>` per impostazione predefinita, è possibile estrarre tale cartella in `C:\` e rinominarla in `elasticsearch` o creare `C:\elasticsearch` autonomamente, quindi decomprimere solo il *contenuto* della cartella nell'archivio a lì.

Poiché Elasticsearch è scritto in Java, ha bisogno di Java Runtime Environment per funzionare. Quindi, prima di eseguire il server, controlla se Java è disponibile aprendo un prompt dei comandi e digitando:

```
java -version
```

Dovresti ricevere una risposta simile a questa:

```
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)
```

Se vedi invece il seguente

> 'java' non è riconosciuto come comando interno o esterno, programma eseguibile o file batch.

Java non è installato sul tuo sistema o non è configurato correttamente. Puoi seguire questo tutorial per (ri) installare Java. Inoltre, assicurati che queste variabili di ambiente siano impostate su valori simili:

Variabile	Valore
JAVA_HOME	C: \ Programmi \ Java \ jre
SENTIERO	...; C: \ Programmi \ Java \ jre

Se non sai ancora come ispezionare queste variabili consulta questo tutorial .

Esegui da un file batch

Con Java installato, apri la cartella `bin` . Può essere trovato direttamente nella cartella in cui è stato decompresso tutto, quindi dovrebbe trovarsi in `c:\elasticsearch\bin` . All'interno di questa cartella si trova un file chiamato `elasticsearch.bat` che può essere utilizzato per avviare Elasticsearch in una finestra di comando. Ciò significa che le informazioni registrate dal processo saranno visibili nella finestra del prompt dei comandi. Per fermare il server, premere `CTRL C` o semplicemente chiudere la finestra.

Esegui come servizio di Windows

Idealmente non si desidera avere una finestra aggiuntiva di cui non è possibile sbarazzarsi durante lo sviluppo e per questo motivo, Elasticsearch può essere configurato per essere eseguito come servizio.

Prima di poter installare Elasticsearch come servizio, è necessario aggiungere una riga al file `C:\elasticsearch\config\jvm.options` :

> Il programma di installazione del servizio richiede che l'impostazione della dimensione dello stack di thread sia configurata in `jvm.options` prima di installare il servizio. Su Windows a 32 bit, dovresti aggiungere `-Xss320k` [...] e su Windows a 64 bit dovresti aggiungere `-Xss1m` al file `jvm.options` . [fonte]

Dopo aver apportato questa modifica, apri un prompt dei comandi e vai alla directory `bin` eseguendo il seguente comando:

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

La gestione dei servizi è gestita da `elasticsearch-service.bat` . Nelle versioni precedenti questo file potrebbe semplicemente essere chiamato `service.bat` . Per vedere tutti gli argomenti disponibili, eseguilo senza:

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

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

L'output ci dice anche che esiste un argomento `SERVICE_ID` facoltativo, ma per il momento possiamo ignorarlo. Per installare il servizio, esegui semplicemente:

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

Dopo aver installato il servizio, puoi avviarlo e interromperlo con i rispettivi argomenti. Per avviare il servizio, esegui

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

e per fermarlo, corri

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

Se preferisci una GUI per gestire il servizio, puoi utilizzare il seguente comando:

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

Verrà aperto Elastic Service Manager, che consente di personalizzare alcune impostazioni relative

ai servizi e di interrompere / avviare il servizio utilizzando i pulsanti disponibili nella parte inferiore della prima scheda.

Indicizzazione e recupero di un documento

Elasticsearch è accessibile tramite un'API REST HTTP, in genere utilizzando la libreria cURL. I messaggi tra il server di ricerca e il client (la tua o la tua applicazione) vengono inviati sotto forma di stringhe JSON. Per impostazione predefinita, Elasticsearch viene eseguito sulla porta 9200.

Negli esempi seguenti, viene aggiunto `?pretty` per dire a Elasticsearch di migliorare la risposta JSON. Quando si utilizzano questi endpoint all'interno di un'applicazione non è necessario aggiungere questo parametro di query.

Documenti di indicizzazione

Se intendiamo aggiornare le informazioni all'interno di un indice in un secondo momento, è una buona idea assegnare ID univoci ai documenti che indicizziamo. Per aggiungere un documento all'indice denominato `megacorp` , con tipo `employee` e ID `1` eseguire:

```
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" ]
}'
```

Risposta:

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

L'indice viene creato se non esiste quando inviamo la chiamata PUT.

Indicizzazione senza ID

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

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

Risposta:

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

Recupero di documenti

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

Risposta:

```
{
  "_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"
    ]
  }
}
```

Scarica 10 documenti dall'indice `megacorp` con il tipo `employee` :

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

Risposta:

```
{
  "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"
          ]
        }
      }
    ]
  }
}
```

Ricerca semplice utilizzando la query di `match` , che cerca corrispondenze esatte nel campo fornito:

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

Risposta:

```
{
  "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"
          ]
        }
      }
    ]
  }
}
```

Parametri di ricerca di base con esempi:

Per impostazione predefinita, il documento indicizzato completo viene restituito come parte di tutte le ricerche. Questo è indicato come fonte (il campo `_source` nei risultati della ricerca). Se non vogliamo che venga restituito l'intero documento sorgente, abbiamo la possibilità di richiedere solo alcuni campi dall'interno della sorgente da restituire, oppure possiamo impostare `_source` su false per omettere completamente il campo.

Questo esempio mostra come restituire due campi, `account_number` e `balance` (all'interno di `_source`), dalla ricerca:

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

Si noti che l'esempio sopra riportato riduce semplicemente le informazioni restituite nel campo `_source` . `_source` comunque solo un campo denominato `_source` ma verranno inclusi solo i campi `account_number` e `balance` .

Se provieni da uno sfondo SQL, quanto sopra è in qualche modo simile nel concetto alla query SQL

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

Passiamo ora alla parte della query. In precedenza, abbiamo visto come viene utilizzata la query `match_all` per abbinare tutti i documenti. Introduciamo ora una nuova query chiamata query di corrispondenza, che può essere pensata come una query di ricerca fielded di base (ovvero una ricerca eseguita su un campo o una serie di campi specifici).

Questo esempio restituisce l'account con `account_number` impostato su `20` :

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

Questo esempio restituisce tutti gli account contenenti il termine "mill" `address` :

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

Questo esempio restituisce tutti gli account contenenti il termine "mill" o "corsia" `address` :

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

Questo esempio è una variante della `match` (`match_phrase`) che divide la query in termini e restituisce solo i documenti che contengono tutti i termini `address` nelle stesse posizioni l'uno rispetto all'altro [1] .

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

Introduciamo ora la query bool (ean). La query bool ci permette di comporre query più piccole in query più grandi usando la logica booleana.

Questo esempio compone due query di corrispondenza e restituisce tutti gli account contenenti "mill" e "corsia" nell'indirizzo:

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

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

Nell'esempio precedente, la clausola bool `must` specificare tutte le query che devono essere vere per un documento da considerare come una corrispondenza.

Al contrario, questo esempio compone due query di corrispondenza e restituisce tutti gli account contenenti "mill" o "corsia" `address` :

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

Nell'esempio precedente, la clausola bool `should` specificare un elenco di query che devono essere vere per un documento da considerare come una corrispondenza.

Questo esempio compone due query di corrispondenza e restituisce tutti gli account che non contengono né "mill" né "corsia" `address` :

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

Nell'esempio precedente, la clausola must_not di bool specifica un elenco di query che nessuna delle quali deve essere vera per un documento considerato come una corrispondenza.

Possiamo combinare le clausole must, should e must_not contemporaneamente all'interno di una query bool. Inoltre, possiamo comporre query bool all'interno di una qualsiasi di queste clausole bool per simulare qualsiasi logica booleana multilivello complessa.

Questo esempio restituisce tutti gli account che appartengono a persone che hanno esattamente 40 anni e che non vivono a Washington (`WA` in breve):

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

Installazione di Elasticsearch e Kibana su CentOS 7

Per eseguire Elasticsearch, è richiesto un Java Runtime Environment (JRE) sulla macchina. Elasticsearch richiede Java 7 o versioni successive e raccomanda `Oracle JDK version 1.8.0_73` .

Quindi, assicurati di avere Java nel tuo sistema. In caso contrario, quindi seguire la procedura:

```
# 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 dovrebbe ora essere installato nel tuo sistema centOS. Puoi verificarlo con:

```
java -version
```

Scarica e installa 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
```

Esecuzione di elasticsearch come servizio systemd all'avvio

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch

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

Installazione di Kibana

Prima importazione chiave GPG su rpm

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

Quindi creare un repository locale `kibana.repo`

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

E aggiungi il seguente contenuto:

```
[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
```

Ora installa il kibana seguendo il seguente comando:

```
yum -y install kibana
```

Inizia con:

```
systemctl start kibana
```

Controlla lo stato con:

```
systemctl status kibana
```

Puoi eseguirlo come servizio di avvio.

```
systemctl enable kibana
```

Leggi Iniziare con Elasticsearch online: https://riptutorial.com/it/elasticsearch/topic/941/iniziare-con-elasticsearch

Capitolo 2: aggregazioni

Sintassi

- "aggregazioni": {- "<aggregation_name>": {- "<aggregation_type>": {- <aggregation_body> -} - [, "meta": {[<meta_data_body>]}]? - [, "aggregazioni": {[<sub_aggregation>] +}]? -} - [, "<aggregation_name_2>": {...}] * -}

Examples

Aggregazione media

Questa è un'aggregazione di metriche a valore singolo che calcola la media dei valori numerici che vengono estratti dai documenti aggregati.

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

L'aggregazione sopra calcola il voto medio su tutti i documenti. Il tipo di aggregazione è avg e l'impostazione del campo definisce il campo numerico dei documenti su cui verrà calcolata la media. Quanto sopra restituirà quanto segue:

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

Il nome dell'aggregazione (avg_grade precedente) funge anche da chiave con la quale il risultato dell'aggregazione può essere recuperato dalla risposta restituita.

Aggregazione di cardinalità

Una aggregazione di metriche a valore singolo che calcola un conteggio approssimativo di valori distinti. I valori possono essere estratti da specifici campi nel documento o generati da uno script.

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

```
                "field" : "type"
            }
        }
    }
}
```

Risposta:

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

Aggregazione di statistiche estesa

Un'aggregazione di metriche multivalore che calcola le statistiche su valori numerici estratti dai documenti aggregati. Questi valori possono essere estratti da specifici campi numerici nei documenti o generati da uno script fornito.

Le aggregazioni extended_stats sono una versione estesa dell'aggregazione delle statistiche, in cui vengono aggiunte metriche aggiuntive come sum_of_squares, variance, std_deviation e std_deviation_bounds.

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

Uscita di esempio:

```
{
    ...

    "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
           }
        }
    }
}
```

Leggi aggregazioni online: https://riptutorial.com/it/elasticsearch/topic/10745/aggregazioni

Capitolo 3: analizzatori

Osservazioni

Gli analizzatori prendono il testo da un campo stringa e generano token che verranno utilizzati durante l'esecuzione di query.

Un analizzatore opera in una sequenza:

- `CharFilters` (zero o più)
- `Tokenizer` (uno)
- `TokenFilters` (zero o più)

L'analizzatore può essere applicato ai mapping in modo che quando i campi vengono indicizzati, viene eseguito su una base di token anziché sulla stringa nel suo complesso. Quando si esegue una query, la stringa di input verrà eseguita anche attraverso l'Analyzer. Pertanto, se normalizzi il testo in Analyzer, esso corrisponderà sempre anche se la query contiene una stringa non normalizzata.

Examples

Mappatura

Un analizzatore può essere applicato a una mappatura utilizzando "analizzatore", per impostazione predefinita viene utilizzato l'analizzatore "standard". In alternativa, se non si desidera utilizzare un analizzatore (poiché la tokenizzazione o la normalizzazione non sarebbe utile) è possibile specificare "index": "not_analyzed"

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

Multi-campi

A volte può essere utile avere più indici distinti di un campo con diversi analizzatori. È possibile

utilizzare la capacità multi-campi per farlo.

```
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"
            }
          }
        }
      }
    }
  }
}
```

Quando si esegue una query, invece di utilizzare semplicemente "user.name" (che in questo caso utilizzerà ancora Stanard Analyzer) è possibile utilizzare "user.name.special" o "user.name.unanalyzed". Si noti che il documento rimarrà invariato, questo influisce solo sull'indicizzazione.

analizzatori

L'analisi in elasticsearch entra nel contesto quando sei disposto ad analizzare i dati nel tuo indice.

Gli analizzatori ci consentono di eseguire quanto segue:

- Abbreviazioni
- Stemming
- Gestione degli errori di battitura

Guarderemo ognuno di loro ora.

1. **Abbreviazioni** :

   Usando gli analizzatori, possiamo dire a elasticsearch come trattare le abbreviazioni nei nostri dati, ad es. Dr => Dottore così quando cerchiamo la parola chiave dottore nel nostro indice, elasticsearch restituirà anche i risultati che hanno menzionato in loro.

2. **Stemming** :

   L'uso dello stemming negli analizzatori ci permette di usare parole base per verbi modificati come

parola	modifiche
richiedere	requisito, richiesto

3. **Gestione degli errori di battitura** :

   Gli analizzatori forniscono anche la gestione degli errori mentre durante l'interrogazione se stiamo cercando una parola specifica che dica "resurrezione", elasticsearch restituirà i risultati in cui sono presenti errori di battitura. Tratterà errori di battitura come resurection, resurrection come lo stesso e riaccorderà il risultato.

parola	modifiche
risurrezione	resurection, ressurection

Analizzatori in Elasticsearch

1. Standard
2. Semplice
3. Lo spazio bianco
4. Stop
5. Parola chiave
6. Modello
7. linguaggio
8. Palla di neve

Ignora l'analizzatore di casi

A volte, potremmo aver bisogno di ignorare il caso della nostra query, in relazione alla corrispondenza nel documento. In questo caso è possibile utilizzare un analizzatore per ignorare il caso durante la ricerca. Ogni campo dovrà contenere questo analizzatore nella sua proprietà, al fine di lavorare:

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

Leggi analizzatori online: https://riptutorial.com/it/elasticsearch/topic/6232/analizzatori

Capitolo 4: API di ricerca

introduzione

L'API di ricerca consente di eseguire una query di ricerca e ottenere risultati di ricerca corrispondenti alla query. La query può essere fornita utilizzando una semplice stringa di query come parametro o utilizzando un corpo della richiesta.

Examples

Routing

Quando si esegue una ricerca, verrà trasmesso a tutti gli indici / indici (round robin tra le repliche). È possibile controllare quali frammenti verranno ricercati fornendo il parametro di instradamento. Ad esempio, durante l'indicizzazione dei tweet, il valore di routing può essere il nome utente:

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

Cerca usando il corpo della richiesta

Le ricerche possono essere eseguite anche su elasticsearch utilizzando un DSL di ricerca. L'elemento di query all'interno del corpo della richiesta di ricerca consente di definire una query utilizzando la query DSL.

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

Ricerca multipla

L'opzione multi_search ci consente di cercare una query in più campi contemporaneamente.

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

```
}
```

Possiamo anche aumentare il punteggio di alcuni campi usando l'operatore boost (^) e usare i caratteri jolly nel nome del campo (*)

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

Ricerca URI e evidenziazione

Una richiesta di ricerca può essere eseguita utilizzando esclusivamente un URI fornendo i parametri di richiesta. Non tutte le opzioni di ricerca vengono esposte quando si esegue una ricerca utilizzando questa modalità, ma può essere utile per eseguire rapidamente "test di arricciatura".

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

Un'altra utile funzione fornita è evidenziare i risultati della partita nei documenti.

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

Nel caso precedente, il campo particolare verrà evidenziato per ogni hit di ricerca

Leggi API di ricerca online: https://riptutorial.com/it/elasticsearch/topic/8625/api-di-ricerca

Capitolo 5: Apprendimento di Elasticsearch con kibana

introduzione

Kibana è uno strumento di visualizzazione dei dati front-end per elasticsearch. per l'installazione di kibana consultare la documentazione di kibana. Per eseguire kibana su localhost, vai su https: // localhost: 5601 e vai alla console di kibana.

Examples

Esplora il tuo Cluster usando Kibana

La sintassi del comando sarà del seguente tipo:

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

Esegui il seguente comando per esplorare il cluster elasticsearch tramite Kibana Console.

- Per verificare la salute del cluster

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

- Per elencare tutti gli indici

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

- Per creare un indice con nome auto

```
 PUT /car?pretty
```

- Per l'indicizzazione del documento con nome auto di tipo esterno utilizzando id 1

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

la risposta della query precedente sarà:

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

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

- il recupero del documento sopra può essere fatto usando:

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

- Per l'eliminazione di un indice

```
DELETE /car?pretty
```

Modifica i tuoi dati di elasticsearch

Elasticsearch offre funzionalità di manipolazione dei dati e ricerca dei dati in tempo quasi reale. sotto questo esempio, abbiamo operazioni di aggiornamento, eliminazione e elaborazione in batch.

- Aggiornamento dello stesso documento. Supponiamo di aver già indicizzato un documento su / car / external / 1. Quindi l'esecuzione del comando per l'indicizzazione dei dati sostituisce il documento precedente.

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

il precedente documento dell'automobile con ID 1 con il nome "Tata Nexon" verrà aggiornato con il nuovo nome "Tata Nexa"

- indicizzazione dei dati con ID esplicito

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

  per l'indicizzazione del documento senza un ID usiamo verbo **POST** anziché verbo **PUT** . se non forniamo un ID, elasticsearch genererà un ID casuale e quindi lo userà per indicizzare il documento.

- Aggiornamento parziale del documento precedente a un ID.

```
POST /car/external/1/_update?pretty
{
```

```
  "doc": { "name": "Tata Nex" }
}
```

- aggiornamento del documento con informazioni aggiuntive

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

- aggiornare il documento usando semplici script.

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

  ctx._source si riferisce al documento sorgente corrente che sta per essere aggiornato. Sopra lo script fornisce solo uno script da aggiornare allo stesso tempo.

- Cancellare il documento

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

  Nota: l'eliminazione di un intero indice è più efficace dell'eliminazione di tutti i documenti utilizzando Elimina per API di query

Elaborazione in lotti

Oltre all'indicizzazione dell'aggiornamento e all'eliminazione del documento, elasticsearch fornisce anche la possibilità di eseguire una delle operazioni sopra elencate in batch utilizzando l'API **_bulk** .

- per l'aggiornamento di più documenti usando **l'** API **_bulk**

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

- per l'aggiornamento e l'eliminazione dei documenti usando **_bulk** API

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

  Se un'operazione fallisce, l'API di massa non si ferma. Esegue tutte le operazioni e infine restituisce il rapporto per tutte le operazioni.

Leggi Apprendimento di Elasticsearch con kibana online:
https://riptutorial.com/it/elasticsearch/topic/10058/apprendimento-di-elasticsearch-con-kibana

Capitolo 6: Configurazione Elasticsearch

Osservazioni

Elasticsearch viene fornito con un set di valori predefiniti che offrono un'esperienza di sviluppo pronta per l'uso. L'affermazione implicita è che non è necessariamente ottimo per la produzione, che deve essere adattato alle proprie esigenze e quindi non può essere previsto.

Le impostazioni predefinite facilitano il download e l'esecuzione di più nodi *sulla stessa macchina* senza modifiche alla configurazione.

Dove sono le impostazioni?

All'interno di ogni installazione di Elasticsearch è un `config/elasticsearch.yml` . È qui che vivono le seguenti impostazioni :

- `cluster.name`
  - Il nome del cluster a cui il nodo si sta unendo. Tutti i nodi nello stesso cluster **devono** condividere lo stesso nome.
  - Attualmente predefinito per `elasticsearch` .
- `node.*`
  - `node.name`
    - Se non viene fornito, verrà generato un nome casuale *ogni volta che il nodo viene avviato* . Questo può essere divertente, ma non è buono per gli ambienti di produzione.
    - I nomi *non* devono essere unici, ma **dovrebbero** essere unici.
  - `node.master`
    - Un'impostazione booleana. Quando è `true` , significa che il nodo è un nodo master idoneo e può essere *il* nodo master eletto.
    - Il valore predefinito è `true` , ovvero ogni nodo è un nodo master idoneo.
  - `node.data`
    - Un'impostazione booleana. Quando è `true` , significa che il nodo memorizza i dati e gestisce l'attività di ricerca.
    - Il valore predefinito è `true` .
- `path.*`
  - `path.data`
    - La posizione in cui i file sono scritti per il nodo. *Tutti i nodi utilizzano questa directory* per archiviare i metadati, ma i nodi dati lo useranno anche per archiviare / indicizzare i documenti.
    - Impostazioni predefinite `./data` .
      - Ciò significa che i `data` verranno creati per te come directory peer da `config` *all'interno* della directory Elasticsearch.
  - `path.logs`
    - La posizione in cui vengono scritti i file di registro.
    - Predefinito `./logs` .
- `network.*`
  - `network.host`

        - Il valore predefinito è `_local_` , che è in effetti `localhost` .
            - Ciò significa che, per impostazione predefinita, i nodi non possono essere comunicati dall'esterno della macchina corrente!

    - `network.bind_host`

        - Potenzialmente un array, questo dice a Elasticsearch quali indirizzi della macchina attuale legano anche i socket.
            - È questo elenco che consente alle macchine esterne alla macchina (ad es. Altri nodi nel cluster) di parlare con questo nodo.
        - Predefinito per `network.host` .

    - `network.publish_host`

        - Un host singolare utilizzato per pubblicizzare su altri nodi come comunicare al meglio con questo nodo.
            - Quando si fornisce un array a `network.bind_host` , questo dovrebbe essere l' *unico* host che si intende utilizzare per la comunicazione tra nodi.
        - Predefinito a network.host`.
- `discovery.zen.*`
    - `discovery.zen.minimum_master_nodes`
        - Definisce il quorum per l'elezione principale. Questo **deve** essere impostato usando questa equazione: `(M / 2) + 1` dove `M` è il numero di nodi principali *eleggibili* (nodi che usano `node.master: true` implicitamente o esplicitamente).
        - Il valore predefinito è `1` , che è valido solo per un cluster a nodo singolo!
    - `discovery.zen.ping.unicast.hosts`
        - Il meccanismo per unire questo nodo al resto di un cluster.
        - Questo *dovrebbe* elencare i nodi master idonei in modo che un nodo possa trovare il resto del cluster.
        - Il valore che dovrebbe essere usato qui è il `network.publish_host` di quegli altri nodi.
        - Il valore predefinito è `localhost` , il che significa che cerca solo sul computer locale per un cluster da unire.

Che tipo di impostazioni esistono?

Elasticsearch offre tre diversi tipi di impostazioni:

- Impostazioni a livello di cluster
    - Queste sono impostazioni valide per tutto il cluster, come tutti i nodi o tutti gli indici.
- Impostazioni del nodo
    - Queste sono impostazioni che si applicano solo al nodo corrente.
- Impostazioni dell'indice
    - Queste sono impostazioni che si applicano solo all'indice.

A seconda dell'impostazione, può essere:

- Modificato dinamicamente in fase di runtime
- Modificato dopo il riavvio (chiusura / apertura) dell'indice

        - Alcune impostazioni a livello di indice non richiedono la chiusura e la riapertura dell'indice, ma potrebbero richiedere l'unione forzata dell'indice per applicare l'impostazione.
            - Il livello di compressione di un indice è un esempio di questo tipo di impostazione. Può essere modificato dinamicamente, ma solo i nuovi *segmenti* approfittano del cambiamento. Quindi, se un indice non cambia, non sfrutta mai la modifica a meno che non si imponga l'indice per ricreare i suoi segmenti.
- Modificato dopo il riavvio del nodo
- Modificato dopo il riavvio del cluster
- Mai cambiato

Controlla sempre la documentazione della tua versione di Elasticsearch per ciò che puoi o non puoi fare con un'impostazione.

Come posso applicare le impostazioni?

Puoi impostare le impostazioni in alcuni modi, alcuni dei quali non sono suggeriti:

- Argomenti della riga di comando

In Elasticsearch 1.xe 2.x, è possibile inviare la maggior parte delle impostazioni come Proprietà di sistema Java con prefisso `es.` :

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

In Elasticsearch 5.x, questo cambia per evitare di utilizzare Java System Properties, invece di usare un tipo di argomento personalizzato con `-E` prendendo il posto di `-Des.` :

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

Questo approccio all'applicazione delle impostazioni funziona alla grande quando si utilizzano strumenti come Puppet, Chef o Ansible per avviare e arrestare il cluster. Tuttavia funziona molto male quando lo fai manualmente.

- Impostazioni YAML
    - Mostrato negli esempi
- Impostazioni dinamiche
    - Mostrato negli esempi

L'ordine in cui vengono applicate le impostazioni è nell'ordine del più dinamico:

1. Impostazioni transitorie
2. Impostazioni persistenti
3. Impostazioni della riga di comando
4. Impostazioni YAML (statiche)

Se l'impostazione è impostata due volte, una volta a uno di questi livelli, il livello più alto ha effetto.

Examples

Impostazioni statiche Elasticsearch

Elasticsearch utilizza un file di configurazione YAML (Yet Another Markup Language) che può essere trovato all'interno della directory Elasticsearch predefinita (le installazioni RPM e DEB cambiano questa posizione tra le altre cose).

Puoi configurare le impostazioni di base in `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_
```

Impostazioni persistenti del cluster dinamico

Se è necessario applicare un'impostazione dinamicamente dopo che il cluster è già stato avviato e può essere impostato in modo dinamico, è possibile impostarlo utilizzando l'API `_cluster/settings` .

Le impostazioni persistenti sono uno dei due tipi di impostazioni a livello di cluster che è possibile applicare. Un'impostazione persistente **sopravviverà** un riavvio completo del cluster.

Nota: non tutte le impostazioni possono essere applicate dinamicamente. Ad esempio, il nome del cluster non può essere rinominato dinamicamente. La maggior parte delle impostazioni a livello di nodo non può essere impostata dinamicamente (perché non possono essere mirate individualmente).

Questa **non** è l'API da utilizzare per impostare le impostazioni a livello di indice. Puoi dire che l'impostazione è un'impostazione di livello indice perché dovrebbe iniziare con `index.` . Impostazioni il cui nome è sotto forma di `indices.` *sono* impostazioni a livello di cluster perché si applicano a tutti gli indici.

```
POST /_cluster/settings
{
  "persistent": {
```

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

Avviso : in Elasticsearch 1.xe 2.x, non è possibile *annullare l'* impostazione persistente.

Fortunatamente, questo è stato migliorato in Elasticsearch 5.x e ora puoi rimuovere un'impostazione impostandola su `null` :

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

Un'impostazione non impostata tornerà al suo valore predefinito o qualsiasi valore definito a un livello di priorità più basso (ad esempio, le impostazioni della riga di comando).

Impostazioni transitorie dinamiche del cluster

Se è necessario applicare un'impostazione dinamicamente dopo che il cluster è già stato avviato e può essere impostato in modo dinamico, è possibile impostarlo utilizzando l'API `_cluster/settings` .

Le impostazioni transitori sono uno dei due tipi di impostazioni a livello di cluster che è possibile applicare. Un'impostazione transitoria **non** sopravviverà al riavvio completo del cluster.

Nota: non tutte le impostazioni possono essere applicate dinamicamente. Ad esempio, il nome del cluster non può essere rinominato dinamicamente. La maggior parte delle impostazioni a livello di nodo non può essere impostata dinamicamente (perché non possono essere mirate individualmente).

Questa **non** è l'API da utilizzare per impostare le impostazioni a livello di indice. Puoi dire che l'impostazione è un'impostazione di livello indice perché dovrebbe iniziare con `index.` . Impostazioni il cui nome è sotto forma di `indices.` *sono* impostazioni a livello di cluster perché si applicano a tutti gli indici.

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

Avviso : in Elasticsearch 1.xe 2.x, non è possibile annullare l'impostazione di una configurazione transitoria senza un riavvio completo del cluster.

Fortunatamente, questo è stato migliorato in Elasticsearch 5.x e ora puoi rimuovere un'impostazione impostandola su null:

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

Un'impostazione non impostata tornerà al suo valore predefinito o qualsiasi valore definito a un livello di priorità più basso (ad esempio, `persistent` impostazioni `persistent`).

Impostazioni dell'indice

Le impostazioni dell'indice sono quelle che si applicano a un singolo indice. Tali impostazioni inizieranno con l' `index.` . L'eccezione a questa regola è `number_of_shards` e `number_of_replicas` , che esistono anche sotto forma di `index.number_of_shards` e `index.number_of_replicas` .

Come suggerisce il nome, le impostazioni a livello di indice si applicano a un singolo indice. Alcune impostazioni devono essere applicate al momento della creazione perché non possono essere modificate dinamicamente, come l'impostazione `index.number_of_shards` , che controlla il numero di shard primari per l'indice.

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

oppure, in un formato più conciso, è possibile combinare prefissi chiave a ciascuno `.` :

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

Gli esempi precedenti creeranno un indice con le impostazioni fornite. È possibile modificare dinamicamente le impostazioni per-index utilizzando l'indice `_settings` endpoint. Ad esempio, qui cambiamo dinamicamente le impostazioni di slowlog *solo* per il livello di avviso:

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

```
  }
}
```

Avviso : Elasticsearch 1.xe 2.x non ha convalidato molto strettamente i nomi delle impostazioni a livello di indice. Se avevi un errore di battitura o semplicemente hai creato un'impostazione, allora l'avrebbe accettata ciecamente, ma altrimenti la ignorerebbe. Elasticsearch 5.x convalida rigorosamente i nomi delle impostazioni e rifiuterà qualsiasi tentativo di applicare le impostazioni dell'indice con una o più impostazioni sconosciute (a causa di un errore di battitura o di un plug-in mancante). Entrambe le istruzioni si applicano alle impostazioni dell'indice che cambiano dinamicamente e al momento della creazione.

Impostazioni dell'indice dinamico per più indici contemporaneamente

Puoi applicare la stessa modifica mostrata nell'esempio `Index Settings` a *tutti* gli indici esistenti con una richiesta, o anche un sottoinsieme di essi:

```
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"
    }
  }
}
```

Se preferisci farlo in modo più selettivo, puoi selezionare multipli senza fornire tutto:

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

Leggi Configurazione Elasticsearch online:
https://riptutorial.com/it/elasticsearch/topic/3411/configurazione-elasticsearch

Capitolo 7: Differenza tra database relazionali ed Elasticsearch

introduzione

Questo è per i lettori che provengono da esperienze relazionali e vogliono imparare elasticsearch. Questo argomento mostra i casi d'uso per i quali i database relazionali non sono un'opzione adatta.

Examples

Differenza terminologica

Database relazionale	elasticsearch
Banca dati	Indice
tavolo	genere
Row / Record	Documento
Nome colonna	campo

Sopra la tabella si disegna approssimativamente un'analogia tra gli elementi di base del database relazionale e elasticsearch.

Impostare

Considerare la struttura seguente in un database relazionale:

```
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
```

Equivalente Elasticsearch:

```
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"
        }
     }
  ]
```

Usi in cui i database relazionali non sono adatti

- L'essenza della ricerca sta nel suo ordine. Tutti vogliono che i risultati della ricerca vengano mostrati in modo tale da mostrare i risultati più adatti. Il database relazionale non ha tale capacità. Elasticsearch d'altra parte mostra i risultati sulla base della pertinenza per impostazione predefinita.

  Impostare

  Come usato nell'esempio precedente.

  Dichiarazione problema

  Supponiamo che l'utente voglia cercare `shirts` ma è interessato alle camicie `red`. In tal caso, i risultati contenenti `red` parola chiave `red` and `shirts` dovrebbero apparire in cima. Poi i risultati per altre magliette dovrebbero essere mostrati dopo di loro.

Soluzione mediante query di database relazionale

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

Produzione

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

Elasticsearch Solution

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

Produzione

```
"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"
     }
  }
 ]
```

Conclusione

Come possiamo vedere sopra, il Database relazionale ha restituito i risultati in ordine casuale, mentre Elasticsearch restituisce risultati in ordine decrescente di `_score` calcolato in base alla pertinenza.

- Tendiamo a commettere errori mentre inseriamo la stringa di ricerca. Ci sono casi in cui

l'utente inserisce un parametro di ricerca errato. I database relazionali non gestiranno tali casi. Elasticsearch in soccorso.

Impostare

Come usato nell'esempio precedente.

Dichiarazione problema

Supponiamo utente vuole cercare `shirts` ma lui entra in una parola non corretta `shrt` per errore. L'utente si aspetta comunque di vedere i risultati della maglietta .

Soluzione mediante query di database relazionale

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

Produzione

```
No results found
```

Elasticsearch Solution

```
POST /test/product/_search

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

Produzione

```
 "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"
    }
  }
]
```

Conclusione

Come possiamo vedere sopra il database relazionale non ha restituito risultati per una parola errata cercata, mentre Elasticsearch che utilizza la sua speciale query `fuzzy` restituisce risultati.

Leggi Differenza tra database relazionali ed Elasticsearch online: https://riptutorial.com/it/elasticsearch/topic/10632/differenza-tra-database-relazionali-ed-elasticsearch

Capitolo 8: Differenza tra indici e tipi

Osservazioni

È facile vedere i `type` come una tabella in un database SQL, in cui l' `index` è il database SQL. Tuttavia, non è un buon modo per avvicinarsi ai `type` .

Tutto sui tipi

In effetti, i tipi sono *letteralmente* solo un campo di metadati aggiunto a ogni documento da Elasticsearch: `_type` . Gli esempi precedenti hanno creato due tipi: `my_type` e `my_other_type` . Ciò significa che ogni documento associato ai tipi ha un campo extra definito automaticamente come `"_type": "my_type"` ; questo è indicizzato con il documento, rendendolo quindi un *campo ricercabile o filtrabile* , ma non ha alcun impatto sul documento stesso, quindi l'applicazione non deve preoccuparsi di ciò.

Tutti i tipi vivono nello stesso indice e quindi negli stessi frammenti collettivi dell'indice. Anche a livello di disco, vivono negli stessi file. L'unica separazione che crea un secondo tipo fornisce è logica. Ogni tipo, sia esso univoco o meno, deve esistere nei mapping e tutti questi mapping devono esistere nello stato del cluster. Questo consuma memoria e, se ogni tipo viene aggiornato dinamicamente, consuma le prestazioni man mano che cambiano i mapping.

Pertanto, è consigliabile definire un solo tipo, a meno che non sia effettivamente necessario un altro tipo. È comune vedere scenari in cui sono desiderabili più tipi. Ad esempio, immagina di avere un indice di auto. Potrebbe essere utile per scomporlo con più tipi:

- BMW
- chevy
- honda
- mazda
- mercedes
- nissan
- Rangerover
- toyota
- ...

In questo modo è possibile cercare tutte le auto o limitarlo per produttore su richiesta. La differenza tra queste due ricerche è semplice come:

```
GET /cars/_search
```

e

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

Ciò che non è ovvio ai nuovi utenti di Elasticsearch è che il secondo modulo è una specializzazione del primo modulo. Viene letteralmente riscritto in:

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

Filtra semplicemente qualsiasi documento che non è stato indicizzato con un campo `_type` cui valore era `bmw` . Poiché ogni documento è indicizzato con il suo tipo come il campo `_type` , questo serve da filtro piuttosto semplice. Se in entrambi gli esempi è stata fornita una ricerca effettiva, il filtro verrà aggiunto alla ricerca completa secondo necessità.

Pertanto, se i tipi sono identici, è molto meglio fornire un singolo tipo (ad esempio, `manufacturer` in questo esempio) ed ignorarlo efficacemente. Quindi, all'interno di ciascun documento, fornisci esplicitamente un campo chiamato `make` *o nome che preferisci* e filtra manualmente su di esso ogni volta che vuoi limitarlo. Questo ridurrà la dimensione dei tuoi mapping a `1/n` dove `n` è il numero di tipi separati. Aggiunge un altro campo a ciascun documento, a vantaggio di una mappatura altrimenti semplificata.

In Elasticsearch 1.xe 2.x, tale campo deve essere definito come

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

1. Il nome è arbitrario.
2. Il nome è arbitrario *e* potrebbe corrispondere al nome del tipo, se lo si desidera anche.

In Elasticsearch 5.x, quanto sopra funzionerà ancora (è deprecato), ma il modo migliore è usare:

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

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

1. Il nome è arbitrario.
2. Il nome è arbitrario *e* potrebbe corrispondere al nome del tipo, se lo si desidera anche.

I tipi dovrebbero essere usati con moderazione all'interno dei vostri indici perché gonfiano le mappature degli indici, di solito senza troppi benefici. Devi averne almeno uno, ma non c'è nulla che dice che devi averne più di uno.

Domande comuni

- Cosa succede se ho due (o più) tipi che sono per lo più identici, ma che hanno alcuni campi univoci per tipo?

A livello di indice, non vi è alcuna differenza tra un tipo usato con pochi campi scarsamente usati *e* tra più tipi che condividono un gruppo di campi non sparsi con pochi non condivisi (il che significa che l'altro tipo non usa mai il campo (S)).

Detto in modo diverso: un campo scarsamente usato è scarso nell'indice *indipendentemente dai tipi* . La sparsità non avvantaggia, o fa davvero male, l'indice solo perché è definito in un tipo separato.

Dovresti semplicemente combinare questi tipi e aggiungere un campo di testo separato.

- Perché i tipi separati devono definire i campi nello stesso modo?

Perché ogni campo viene definito solo una volta a livello di Lucene, indipendentemente da quanti tipi ci sono. Il fatto che i tipi esistano è una caratteristica di Elasticsearch ed è *solo* una separazione logica.

- Posso definire tipi separati con lo stesso campo definito in modo diverso?

No. Se riesci a trovare un modo per farlo in ES 2.xo versioni successive, dovresti aprire una segnalazione di bug . Come notato nella domanda precedente, Lucene li vede tutti come un singolo campo, quindi non c'è modo di fare questo lavoro in modo appropriato.

ES 1.x lo ha lasciato come requisito implicito, che ha consentito agli utenti di creare condizioni in cui le mappature di un frammento in un indice in realtà differivano da un altro frammento nello stesso indice. Questa era effettivamente una condizione di gara e *poteva* portare a problemi imprevisti.

Eccezioni alla regola

- I documenti padre / figlio **richiedono** tipi separati da utilizzare all'interno dello stesso indice.
    - Il genitore vive in un tipo.

    - Il bambino vive in un tipo separato (ma ogni bambino vive nella stessa *scheggia del* genitore).
- Casi di utilizzo estremamente di nicchia in cui creare tonnellate di indici è indesiderabile e l'impatto dei campi sparsi è preferibile all'alternativa.
    - Ad esempio, il plug-in di monitoraggio Elasticsearch, Marvel (1.xe 2.x) o X-Pack Monitoring (5.x +), monitora Elasticsearch stessa per le modifiche nel cluster, i nodi, gli indici, gli indici specifici (il livello dell'indice), e anche frammenti. Potrebbe creare più di 5+ indici ogni giorno per isolare quei documenti che hanno mappature univoche *o* potrebbe andare contro le migliori pratiche per ridurre il carico del cluster condividendo un indice (nota: il numero di mappature definite è effettivamente lo stesso, ma il numero di indici creati è ridotto da `n` a 1).
    - Questo è uno scenario avanzato, ma è necessario considerare le definizioni dei campi condivise tra i tipi!

Examples

Creazione esplicita di un indice con un tipo

Esempio utilizza HTTP di base, che si traduce facilmente in cURL e altre applicazioni HTTP. Corrispondono anche alla sintassi Sense , che verrà rinominata Console in Kibana 5.0.

Nota: l'esempio inserisce `<#>` per attirare l'attenzione sulle parti. Quelli dovrebbero essere rimossi se lo copi!

```
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. Questo sta creando l' `index` usando l'endpoint di creazione dell'indice.
2. Questo sta creando il `type` .
3. I campi condivisi in `type` s all'interno dello stesso `index` **devono** condividere la stessa definizione! ES 1.x non ha applicato rigorosamente questo comportamento, ma era un requisito implicito. ES 2.xe precedenti applicano rigorosamente questo comportamento.
4. I campi unici tra i `type` s sono a posto.

Gli indici (o indici) *contengono* tipi. I tipi sono un meccanismo utile per separare i documenti, ma richiedono di definire, dinamicamente / automaticamente o esplicitamente, una mappatura per ogni tipo che si utilizza. Se si definiscono 15 tipi in un indice, si avranno 15 mappature univoche.

Vedi le osservazioni per maggiori dettagli su questo concetto e perché potresti o non vuoi usare i tipi.

Creazione dinamica di un indice con un tipo

Esempio utilizza HTTP di base, che si traduce facilmente in cURL e altre applicazioni HTTP. Corrispondono anche alla sintassi Sense , che verrà rinominata Console in Kibana 5.0.

Nota: l'esempio inserisce `<#>` per attirare l'attenzione sulle parti. Quelli dovrebbero essere rimossi se lo copi!

```
DELETE /my_index <1>

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

1. Nel caso esista già (a causa di un esempio precedente), eliminare l'indice.
2. Indicizza un documento nell'indice, `my_index` , con il tipo, `my_type` e l'ID `abc123` (potrebbe essere numerico, ma è sempre una stringa).
    - Per impostazione predefinita, la creazione dell'indice dinamico è abilitata semplicemente indicizzando un documento. Questo è ottimo per gli ambienti di sviluppo, ma non è necessariamente un bene per gli ambienti di produzione.
3. Questo campo è un numero intero, quindi la prima volta che viene visto deve essere mappato. Elasticsearch assume sempre il tipo *più largo* per qualsiasi tipo di ingresso, quindi questo verrà mappato come un numero `long` anziché un `integer` o un `short` (entrambi potrebbero contenere `1234` e `456`).
4. Lo stesso vale per questo campo. Sarà mappato come un `double` invece di un `float` come si potrebbe desiderare.

Questo indice e tipo creati dinamicamente corrispondono alla mappatura definita nel primo esempio. Tuttavia, è fondamentale capire in che modo `<3>` e `<4>` influiscono sui mapping definiti automaticamente.

Puoi seguire questo aggiungendo un altro tipo in modo dinamico allo stesso indice:

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

1. Il tipo è l'unica differenza dal documento sopra. L'ID è lo stesso e va bene! Non ha alcuna relazione con l'altro `abc123` oltre a quello che si *trova* nello stesso indice.
2. `field1` esiste già nell'indice, quindi *deve* essere lo stesso tipo di campo definito negli altri tipi. L'invio di un valore che fosse una stringa o non un intero sarebbe fallito (ad es. `"field1": "this is some text"` o `"field1": 123.0`).

Ciò crea dinamicamente i mapping per `my_other_type` all'interno dello stesso indice, `my_index` .

Nota: è *sempre* più rapido definire le associazioni in anticipo anziché avere Elasticsearch eseguirlo dinamicamente al momento dell'indice.

Il risultato finale dell'indicizzazione di entrambi i documenti sarebbe simile al primo esempio, ma i tipi di campo sarebbero diversi e quindi leggermente dispendiosi:

```
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. Questo utilizza l'endpoint `_mappings` per ottenere i mapping dall'indice che abbiamo creato.
2. Abbiamo creato dinamicamente `my_type` nel primo passaggio di questo esempio.
3. `field2` è ora un `long` invece di un `integer` perché non lo abbiamo definito in anticipo. Questo *potrebbe* rivelarsi uno spreco nella memoria del disco.
4. `object1.field1` ora è un `double` per la stessa ragione del # 3 con le stesse ramificazioni del # 3.
    - Tecnicamente, un `long` può essere compresso in molti casi. Tuttavia, un `double` non può essere compresso perché è un numero in virgola mobile.
5. Abbiamo anche creato dinamicamente `my_other_type` nel secondo passaggio di questo esempio. La sua mappatura sembra essere la stessa perché stavamo già usando `long` e `double` .
    - Ricorda che `field1` *deve* corrispondere alla definizione di `my_type` (e lo fa).
    - `field3` è unico per questo tipo, quindi non ha tale restrizione.

Leggi Differenza tra indici e tipi online: https://riptutorial.com/it/elasticsearch/topic/3412/differenza-tra-indici-e-tipi

Capitolo 9: Grappolo

Osservazioni

Cluster Health fornisce molte informazioni sul cluster, ad esempio il numero di frammenti allocati ("attivi") e il numero di assegnazioni non assegnate e di riposizionamento. Inoltre, fornisce il numero corrente di nodi e nodi dati nel cluster, che può consentire di eseguire il polling per i nodi mancanti (ad esempio, se ci si aspetta che sia `15` , ma mostra solo `14` , allora manca un nodo) .

Per qualcuno che conosce Elasticsearch, i frammenti "assegnati" e "non assegnati" possono aiutarli a rintracciare i problemi.

Il campo più comune verificato da Cluster Health è lo `status` , che può essere in uno dei tre stati:

- rosso
- giallo
- verde

I colori significano ognuno - e solo uno - una cosa molto semplice:

1. Il rosso indica che ti manca *almeno* un frammento primario.
    - Un frammento primario mancante indica che un indice non può essere utilizzato per scrivere (indicizzare) nuovi dati nella maggior parte dei casi.
        - Tecnicamente, puoi comunque indicizzare qualsiasi frammento primario disponibile in quell'indice, ma in pratica significa che non puoi farlo perché generalmente non controlli quale frammento riceve un determinato documento.
        - La ricerca è ancora possibile contro un cluster rosso, ma significa che otterrai risultati parziali se qualsiasi indice che cerchi manca di frammenti.
    - In circostanze normali, significa solo che il frammento primario è stato allocato (`initializing_shards`).
    - Se un nodo ha appena lasciato il cluster (ad esempio, poiché la macchina è in esecuzione ha perso potenza), allora ha senso che mancherà *temporaneamente* alcuni frammenti primari.
        - Qualsiasi frammento di replica per quel frammento primario verrà promosso come frammento primario in questo scenario.
2. Il giallo indica che tutti i frammenti primari sono attivi, ma manca *almeno* un frammento di replica.
    - Una replica mancante influisce solo sull'indicizzazione se le impostazioni di coerenza richiedono un impatto sull'indicizzazione.
        - Per impostazione predefinita, esiste una sola replica per qualsiasi primario e l'indicizzazione può avvenire con una singola replica mancante.
    - In circostanze normali, significa solo che il frammento di replica è stato allocato (`initializing_shards`).
    - Un cluster con un nodo con repliche abilitate sarà *sempre* giallo *al meglio* . Può essere rosso se non è stato ancora assegnato un frammento primario.
        - Se si ha un solo nodo, allora ha senso disabilitare le repliche perché non le si

prevedono. Quindi può essere verde.

3. Il verde indica che tutti i frammenti sono attivi.
    - L'unica attività di shard consentita per un cluster verde è `relocating_shards` .
    - Nuovi indici, e quindi nuovi frammenti, faranno passare il cluster da rosso a giallo a verde, dato che ogni frammento viene allocato (primariamente per primo, rendendolo giallo, quindi repliche se possibile, rendendolo verde).
        - In Elasticsearch 5.xe versioni successive, i nuovi indici **non** renderanno il cluster rosso a meno che non impieghino troppo tempo per essere allocati.

Examples

Cruscotto umano leggibile, tabulare con intestazioni

Esempio utilizza la sintassi HTTP di base. Qualsiasi `<#>` nell'esempio dovrebbe essere rimosso quando lo si copia.

È possibile utilizzare le API `_cat` per ottenere un output tabulare leggibile dall'uomo per vari motivi.

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

1. Il `?v` è facoltativo, ma implica che si desidera un output "dettagliato".

`_cat/health` esiste da Elasticsearch 1.x, ma ecco un esempio del suo output da Elasticsearch 5.x:

Con output dettagliato:

```
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%
```

Cruscotto umano leggibile, tabulare senza intestazioni

Esempio utilizza la sintassi HTTP di base. Qualsiasi `<#>` nell'esempio dovrebbe essere rimosso quando lo si copia.

È possibile utilizzare le API `_cat` per ottenere un output tabulare leggibile dall'uomo per vari motivi.

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

`_cat/health` esiste da Elasticsearch 1.x, ma ecco un esempio del suo output da Elasticsearch 5.x:

Senza output dettagliato:

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

Cruscotto umano leggibile, tabulare con intestazioni selezionate

Esempio utilizza la sintassi HTTP di base. Qualsiasi `<#>` nell'esempio dovrebbe essere rimosso quando lo si copia.

Come la maggior `_cat` API `_cat` in Elasticsearch, l'API risponde selettivamente con un set predefinito di campi. Tuttavia, altri campi esistono dall'API se li vuoi:

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

1. `?help` fa sì che l'API restituisca i campi (e nomi brevi) e una breve descrizione.

`_cat/health` esiste da Elasticsearch 1.x, ma ecco un esempio del suo output da Elasticsearch 5.x:

Campi disponibili come: della data di creazione di questo esempio:

```
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
```

Puoi quindi usarlo per stampare solo quei campi:

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

1. `h=...` definisce l'elenco dei campi che vuoi vengano restituiti.
2. `v` (verbose) definisce che si desidera che stampi le intestazioni.

L'output di un'istanza di Elasticsearch 5.x:

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

Cluster Health basato su JSON

Esempio utilizza la sintassi HTTP di base. Qualsiasi `<#>` nell'esempio dovrebbe essere rimosso quando lo si copia.

Le API `_cat` sono spesso convenienti per gli utenti per ottenere dettagli a colpo d'occhio sul cluster. Ma spesso si desidera un output costantemente parsabile da utilizzare con il software. In generale, le API JSON sono pensate per questo scopo.

```
GET /_cluster/health
```

`_cluster/health` esiste da Elasticsearch 1.x, ma ecco un esempio del suo output da 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
}
```

Leggi Grappolo online: https://riptutorial.com/it/elasticsearch/topic/2069/grappolo

Capitolo 10: Interfaccia Python

Parametri

Parametro	Dettagli
padroni di casa	Matrice di host sotto forma di oggetto contenente chiavi `host` e `port` . L' `host` predefinito è 'localhost' e la `port` è 9200. Una voce di esempio assomiglia a `[{"host": "ip of es server", "port": 9200}]`
sniff_on_start	Booleano se si desidera che il client annodi i nodi all'avvio, sniffing significa ottenere l'elenco dei nodi nel cluster elasticsearch
sniff_on_connection_fail	Booleano per l'attivazione dello sniffing se la connessione non riesce quando il client è attivo
sniffer_timeout	differenza di tempo in secondi tra ogni annusata
sniff_timeout	tempo per una singola richiesta di sniffing in pochi secondi
retry_on_timeout	Booelan per se il client dovrebbe timeout innescare contattando un diverso nodo elasticsearch o semplicemente lanciare un errore
http_auth	L'autenticazione http di base può essere fornita qui sotto forma di `username:password`

Examples

Indicizzazione di un documento (ad es. Aggiunta di un campione)

Installa la libreria Python necessaria tramite:

```
$ pip install elasticsearch
```

Collegati a Elasticsearch, crea un documento (ad es. Inserimento dati) e "indicizza" il documento utilizzando 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"])
```

Connessione 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)
```

Creazione di un indice vuoto e impostazione della mappatura

In questo esempio, creiamo un indice vuoto (non indichiamo alcun documento in esso) definendone la mappatura.

Per prima cosa, creiamo un'istanza di `ElasticSearch` e quindi definiamo la mappatura della nostra scelta. Successivamente, controlliamo se l'indice esiste e, in caso contrario, lo creiamo specificando i parametri `index` e del `body` che contengono rispettivamente il nome dell'indice e il corpo della mappatura.

```
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))
```

Aggiornamento e aggiornamento parziale tramite query

Aggiornamento parziale: utilizzato quando è necessario un aggiornamento parziale del documento, ovvero nell'esempio seguente il `name` del campo del documento con id `doc_id` verrà aggiornato in "John". Notare che se il campo è mancante, verrà semplicemente aggiunto al documento.

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

Aggiorna per query: utilizzata quando è necessario aggiornare i documenti che soddisfano una condizione, ovvero nell'esempio seguente viene aggiornata l'età dei documenti il cui campo del `name` corrisponde a "Giovanni".

```
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')
```

Leggi Interfaccia Python online: https://riptutorial.com/it/elasticsearch/topic/2068/interfaccia-python

Capitolo 11: Riccioli

Sintassi

- curl -X <VERB> '<PROTOCOL>: // <HOST>: <PORT> / <PATH>? <QUERY_STRING>' -d '<BODY>'
- Dove:
- VERBO: il metodo o il verbo HTTP appropriato: GET, POST, PUT, HEAD o DELETE
- PROTOCOLLO: http o https (se hai un proxy https davanti a Elasticsearch).
- HOST: il nome host di qualsiasi nodo nel cluster Elasticsearch o localhost per un nodo sul computer locale.
- PORT: la porta su cui è in esecuzione il servizio HTTP Elasticsearch, che per impostazione predefinita è 9200.
- PERCORSO: Endpoint API (ad esempio, _count restituirà il numero di documenti nel cluster). Il percorso può contenere più componenti, come _cluster / stats o _nodes / stats / jvm
- QUERY_STRING: qualsiasi parametro di stringa di query facoltativo (ad esempio? Pretty stamperà la risposta JSON per renderlo più facile da leggere).
- BODY: un corpo di richiesta con codifica JSON (se la richiesta ne richiede uno).
- Riferimento: Talking to Elasticsearch: Elasticsearch Docs

Examples

Ricciolo Comando per il conteggio del numero di documenti nel cluster

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

Produzione:

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

L'indice ha 90 documenti al suo interno.

Link di riferimento: qui

Recupera un documento con Id

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

Produzione:

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

Link di riferimento: qui

Crea un indice

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

Produzione:

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

Link di riferimento: qui

Elenca tutti gli indici

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

produzione:

```
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
```

Link di riferimento: qui

Elimina un indice

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

produzione:

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

Link di riferimento: qui

Elenca tutti i documenti in un indice

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

Questo utilizza l'API di `Search` e restituirà tutte le voci sotto index `myIndexName` .

Link di riferimento: qui

Leggi Riccioli online: https://riptutorial.com/it/elasticsearch/topic/3703/riccioli

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare 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	aggregazioni	Sid1199
3	analizzatori	Bhushan Gadekar, Sid1199, Thomas
4	API di ricerca	aerokite, Sid1199
5	Apprendimento di Elasticsearch con kibana	sarvajeetsuman
6	Configurazione Elasticsearch	pickypg
7	Differenza tra database relazionali ed Elasticsearch	Richa
8	Differenza tra indici e tipi	pickypg
9	Grappolo	Gerardo Rochín, pickypg
10	Interfaccia Python	aidan.plenert.macdonald, christinabo, KartikKannapur, pickypg, Sumit Kumar
11	Riccioli	Fawix, Mrunal Pagnis, Mrunal Pagnis