Kostenloses eBook

LERNEN

Elasticsearch

Free unaffiliated eBook created from
Stack Overflow contributors.

#elasticsear

ch

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit Elasticsearch

Bemerkungen

Elasticsearch ist ein erweiterter Open-Source-Suchserver, der auf Lucene basiert und in Java geschrieben ist.

Sie bietet verteilte, abfrage- und geolokationsbasierte Suchfunktionen für vollständigen und unvollständigen Text, auf die über eine HTTP-REST-API zugegriffen werden kann.

Versionen

Ausführung	Veröffentlichungsdatum
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

Ausführung	Veröffentlichungsdatum
1.2.0	2014-05-22
1.1.0	2014-03-25
1.0.0	2014-02-14

Examples

Installation von Elasticsearch unter Ubuntu 14.04

Voraussetzungen

Zur Ausführung von Elasticsearch ist auf dem Computer eine Java Runtime Environment (JRE) erforderlich. Elasticsearch erfordert Java 7 oder höher und empfiehlt `Oracle JDK version 1.8.0_73` .

Installieren Sie 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
```

Überprüfen Sie die Java-Version

```
java -version
```

Paket herunterladen und installieren

Verwenden von Binärdateien

1. Laden Sie die neueste stabile Version von Elasticsearch hier herunter.
2. Entpacken Sie die Datei & Ausführen

Linux:

```
$ bin/elasticsearch
```

Mit apt-get

Eine Alternative zum Herunterladen von elasticsearch von der Website ist die Installation mit `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
```

Elasticsearch Version 5.x installieren

```
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
```

Läuft als Dienst unter Linux:

Nach der Installation startet das obige nicht von selbst. Wir müssen es also als Dienstleistung starten. Wie Sie Elasticsearch starten oder stoppen, hängt davon ab, ob Ihr System SysV init oder systemd verwendet. Sie können es mit dem folgenden Befehl überprüfen.

```
ps -p 1
```

Wenn Ihre Distribution SysV init verwendet, müssen Sie Folgendes ausführen:

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

Ansonsten, wenn Ihre Distribution systemd verwendet:

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

Führen Sie den Befehl `CURL` Ihrem Browser oder einem REST-Client aus, um zu prüfen, ob Elasticsearch ordnungsgemäß installiert wurde.

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

Installation von Elasticsearch unter Windows

Voraussetzungen

Die Windows-Version von Elasticsearch kann über diesen Link abgerufen werden: https://www.elastic.co/downloads/elasticsearch . Die neueste stabile Version ist immer ganz oben.

Bei der Installation unter Windows benötigen wir das `.ZIP` Archiv. Klicken Sie auf den Link im

Abschnitt `Downloads:` und speichern Sie die Datei auf Ihrem Computer.

Diese Version von Elastic ist "tragbar", dh Sie müssen kein Installationsprogramm ausführen, um das Programm verwenden zu können. Entpacken Sie den Inhalt der Datei an einen Ort, an den Sie sich leicht erinnern können. Zur Demonstration nehmen wir an, dass Sie alles nach `C:\elasticsearch` elasticsearch entpackt haben.

Beachten Sie, dass das Archiv standardmäßig einen Ordner namens `elasticsearch-<version>` enthält. Sie können diesen Ordner entweder nach `C:\` extrahieren und in `elasticsearch` umbenennen oder `C:\elasticsearch` selbst erstellen und dann nur den *Inhalt* des Ordners im Archiv entpacken dorthin

Da Elasticsearch in Java geschrieben ist, muss Java Runtime Environment verwendet werden. Überprüfen Sie daher vor dem Ausführen des Servers, ob Java verfügbar ist, indem Sie eine Eingabeaufforderung öffnen und Folgendes eingeben:

```
java -version
```

Sie sollten eine Antwort erhalten, die so aussieht:

```
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)
```

Wenn Sie stattdessen Folgendes sehen

> 'java' wird nicht als interner oder externer Befehl, bedienbares Programm oder Batchdatei erkannt.

Java ist nicht auf Ihrem System installiert oder nicht ordnungsgemäß konfiguriert. Sie können diesem Tutorial folgen, um Java (neu) zu installieren. Stellen Sie außerdem sicher, dass diese Umgebungsvariablen auf ähnliche Werte gesetzt sind:

Variable	Wert
JAVA_HOME	C: \ Programme \ Java \ jre
PFAD	…; C: \ Programme \ Java \ jre

Wenn Sie noch nicht wissen, wie diese Variablen untersucht werden sollen, lesen Sie dieses Tutorial .

Führen Sie die Batchdatei aus

Öffnen Sie bei installiertem Java den Ordner `bin` . Es befindet sich direkt in dem Ordner, in den Sie alles entpackt haben, also unter `c:\elasticsearch\bin` . In diesem Ordner befindet sich eine Datei namens `elasticsearch.bat` , mit der Elasticsearch in einem Befehlsfenster gestartet werden

kann. Dies bedeutet, dass vom Prozess protokollierte Informationen im Eingabeaufforderungsfenster angezeigt werden. Um den Server zu stoppen, drücken Sie `STRG C` oder schließen Sie einfach das Fenster.

Als Windows-Dienst ausführen

Idealerweise möchten Sie kein zusätzliches Fenster haben, das Sie während der Entwicklung nicht loswerden können. Aus diesem Grund kann Elasticsearch so konfiguriert werden, dass es als Dienst ausgeführt wird.

Bevor wir Elasticsearch als Dienst installieren können, müssen Sie der Datei `C:\elasticsearch\config\jvm.options` eine Zeile `C:\elasticsearch\config\jvm.options` :

> Das Service-Installationsprogramm erfordert, dass die Einstellung für die Thread- `jvm.options` in `jvm.options` konfiguriert `jvm.options` bevor Sie den Service installieren. Unter 32-Bit-Windows sollten Sie `-Xss320k` […] und unter 64-Bit-Windows `-Xss1m` der Datei `-Xss1m` `jvm.options` . [Quelle]

Wenn Sie diese Änderung vorgenommen haben, öffnen Sie eine Eingabeaufforderung und navigieren Sie mit dem folgenden Befehl zum `bin` Verzeichnis:

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

Das Service Management wird von `elasticsearch-service.bat` abgewickelt. In älteren Versionen könnte diese Datei einfach als `service.bat` . Um alle verfügbaren Argumente anzuzeigen, führen Sie sie ohne aus:

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

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

Die Ausgabe sagt uns auch, dass es ein optionales `SERVICE_ID` Argument gibt, wir können es jedoch `SERVICE_ID` ignorieren. Um den Dienst zu installieren, führen Sie einfach Folgendes aus:

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

Nach der Installation des Dienstes können Sie ihn mit den entsprechenden Argumenten starten und stoppen. Starten Sie den Dienst, um den Dienst zu starten

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

und um es zu stoppen, renne

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

Wenn Sie stattdessen eine GUI zur Verwaltung des Dienstes bevorzugen, können Sie den

folgenden Befehl verwenden:

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

Daraufhin wird der Elastic Service Manager geöffnet, in dem Sie einige dienstbezogene Einstellungen anpassen und den Dienst mithilfe der Schaltflächen am unteren Rand der ersten Registerkarte anhalten bzw. starten können.

Dokument indizieren und abrufen

Der Zugriff auf Elasticsearch erfolgt über eine HTTP-REST-API, normalerweise unter Verwendung der cURL-Bibliothek. Die Nachrichten zwischen dem Suchserver und dem Client (Ihrer oder Ihrer Anwendung) werden in Form von JSON-Zeichenfolgen gesendet. Standardmäßig wird Elasticsearch auf Port 9200 ausgeführt.

In den folgenden Beispielen wird `?pretty` Pretty hinzugefügt, um Elasticsearch anzuweisen, die JSON-Antwort zu verbessern. Wenn Sie diese Endpunkte innerhalb einer Anwendung verwenden, müssen Sie diesen Abfrageparameter nicht hinzufügen.

Dokumente indizieren

Wenn wir Informationen innerhalb eines Index später aktualisieren möchten, empfiehlt es sich, den von uns indizierten Dokumenten eindeutige IDs zuzuweisen. So fügen Sie ein Dokument mit dem Namen `megacorp` , mit dem Typ `employee` und 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" ]
}'
```

Antwort:

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

Der Index wird erstellt, wenn er beim Senden des PUT-Aufrufs nicht vorhanden ist.

Indizierung ohne ID

```
POST /megacorp/employee?pretty
{
    "first_name" :  "Jane",
    "last_name" :   "Smith",
    "age" :         32,
    "about" :       "I like to collect rock albums",
    "interests":  [ "music" ]
}
```

Antwort:

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

Dokumente abrufen

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

Antwort:

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

Holen Sie 10 Dokumente aus dem `megacorp` Index mit dem Typ `employee` :

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

Antwort:

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

Einfache Suche mit der `match` , die im angegebenen Feld nach genauen Übereinstimmungen sucht:

```
curl -XGET "http://localhost:9200/megacorp/employee/_search" -d'
```

```
{
    "query" : {
        "match" : {
            "last_name" : "Smith"
        }
    }
}'
```

Antwort:

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

Grundlegende Suchparameter mit Beispielen:

Standardmäßig wird das vollständig indizierte Dokument als Teil aller Suchvorgänge zurückgegeben. Dies wird als Quelle (Feld `_source` in den `_source`) bezeichnet. Wenn wir nicht möchten, dass das gesamte Quelldokument zurückgegeben wird, haben wir die Möglichkeit, nur einige wenige Felder aus der Quelle `_source` , oder `_source` auf false gesetzt werden, um das Feld vollständig auszulassen.

Dieses Beispiel zeigt, wie zwei Felder, `account_number` und `balance` (innerhalb von `_source`) aus der Suche zurückgegeben werden:

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

```
  "query": { "match_all": {} },
  "_source": ["account_number", "balance"]
}'
```

Beachten Sie, dass das obige Beispiel die im Feld `_source` zurückgegebenen Informationen reduziert. Es wird immer nur ein Feld mit dem Namen `_source` jedoch werden nur die Felder `account_number` und `balance` berücksichtigt.

Wenn Sie aus einem SQL-Hintergrund stammen, ähnelt das obige Konzept der SQL-Abfrage

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

Nun geht es weiter zum Abfrageteil. Bisher haben wir gesehen, wie die `match_all` Abfrage verwendet wird, um alle Dokumente `match_all` . Lassen Sie uns nun eine neue Abfrage einführen, die Übereinstimmungsabfrage genannt wird. Diese Abfrage kann als einfache Feldsuchabfrage (z. B. Suche nach einem bestimmten Feld oder einer Gruppe von Feldern) betrachtet werden.

In diesem Beispiel wird das Konto mit der `account_number 20` :

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

In diesem Beispiel werden alle Konten zurückgegeben, die den Begriff "Mühle" in der `address` :

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

In diesem Beispiel werden alle Konten zurückgegeben, die den Begriff "Mühle" oder "Spur" in der `address` :

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

Dieses Beispiel ist eine Variante von `match` (`match_phrase`), die die Abfrage in Begriffe `match_phrase` und nur Dokumente zurückgibt, die alle Begriffe in der `address` relativ zueinander an denselben Positionen enthalten [1] .

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

Lassen Sie uns nun die Abfrage bool (ean) einführen. Die bool-Abfrage ermöglicht es uns, kleinere Abfragen mithilfe der booleschen Logik zu größeren Abfragen zusammenzusetzen.

In diesem Beispiel werden zwei Übereinstimmungsabfragen erstellt und alle Konten zurückgegeben, die "mill" und "lane" in der Adresse enthalten:

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

Im obigen Beispiel gibt die Klausel bool `must` alle Abfragen an, die erfüllt sein müssen, damit ein Dokument als Übereinstimmung betrachtet wird.

Im Gegensatz dazu erstellt dieses Beispiel zwei Übereinstimmungsabfragen und gibt alle Konten zurück, die "mill" oder "lane" in der `address` :

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

Im obigen Beispiel gibt die Klausel bool `should` eine Liste von Abfragen an, die beide zutreffen müssen, damit ein Dokument als Übereinstimmung betrachtet wird.

In diesem Beispiel werden zwei Übereinstimmungsabfragen erstellt und alle Konten zurückgegeben, die weder "mill" noch "lane" in der `address` :

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

Im obigen Beispiel gibt die bool-Klausel must_not eine Liste von Abfragen an, von denen keine wahr sein muss, damit ein Dokument als Übereinstimmung betrachtet wird.

Innerhalb einer bool-Abfrage können wir must-, soll- und must_not-Klauseln gleichzeitig kombinieren. Darüber hinaus können Bool-Abfragen in einer dieser Bool-Klauseln zusammengestellt werden, um komplexe Boolesche Logik auf mehreren Ebenen nachzuahmen.

In diesem Beispiel werden alle Konten zurückgegeben, die Personen gehören, die genau 40 Jahre alt sind und nicht in Washington (kurz `WA`) leben:

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

Elasticsearch und Kibana unter CentOS 7 installieren

Zur Ausführung von Elasticsearch ist auf dem Computer eine Java Runtime Environment (JRE) erforderlich. Elasticsearch erfordert Java 7 oder höher und empfiehlt `Oracle JDK version 1.8.0_73` .

Stellen Sie also sicher, dass Sie Java in Ihrem System haben. Wenn nicht, folgen Sie den Anweisungen:

```
# 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 sollte jetzt in Ihrem centOS-System installiert sein. Sie können es überprüfen mit:

```
java -version
```

Elasticsearch herunterladen und installieren

```
# 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
```

Ausführen von elasticsearch als systemd-Dienst beim Start

```
sudo systemctl daemon-reload
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch

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

Kibana installieren

Importiere zuerst den GPG-Schlüssel bei rpm

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

Dann erstellen Sie ein lokales Repository `kibana.repo`

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

Und füge folgenden Inhalt hinzu:

```
[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
```

Installieren Sie nun die Kibana mit folgendem Befehl:

```
yum -y install kibana
```

Beginnen Sie mit:

```
systemctl start kibana
```

Status prüfen mit:

```
systemctl status kibana
```

Sie können es als Startdienst ausführen.

```
systemctl enable kibana
```

Erste Schritte mit Elasticsearch online lesen:
https://riptutorial.com/de/elasticsearch/topic/941/erste-schritte-mit-elasticsearch

Kapitel 2: Aggregationen

Syntax

- "aggregations": {- "<aggregation_name>": {- "<aggregation_type>": {- <aggregation_body> -} - [, "meta": {[<meta_data_body>]]? - [, "Aggregationen": {[<sub_aggregation>] +}]? -} - [, "<Aggregationsname_2>": {...}] * -}

Examples

Durchschnittliche Aggregation

Dies ist eine Aggregation mit Einzelwertmetriken, die den Durchschnitt der numerischen Werte berechnet, die aus den aggregierten Dokumenten extrahiert werden.

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

Die obige Aggregation berechnet die Durchschnittsnote aller Dokumente. Der Aggregationstyp ist avg und die Feldeinstellung definiert das numerische Feld der Dokumente, für die der Durchschnitt berechnet wird. Das Obige wird Folgendes zurückgeben:

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

Der Name der Aggregation (oben avg_grade) dient auch als Schlüssel, mit dem das Aggregationsergebnis aus der zurückgegebenen Antwort abgerufen werden kann.

Kardinalitätsaggregation

Eine Einzelwertmetrikaggregation, die eine ungefähre Anzahl von unterschiedlichen Werten berechnet. Werte können entweder aus bestimmten Feldern des Dokuments extrahiert oder von einem Skript generiert werden.

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

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

Antwort:

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

Erweiterte Statistikaggregation

Eine Multi-Value-Metrik-Aggregation, die Statistiken über numerische Werte berechnet, die aus den aggregierten Dokumenten extrahiert werden. Diese Werte können entweder aus bestimmten numerischen Feldern in den Dokumenten extrahiert oder von einem bereitgestellten Skript generiert werden.

Die extended_stats-Aggregation ist eine erweiterte Version der stats-Aggregation, in der zusätzliche Metriken wie sum_of_squares, varance, std_deviation und std_deviation_bounds hinzugefügt werden.

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

Beispielausgabe:

```
{
    ...

    "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
           }
```

```
            }
        }
}
```

Aggregationen online lesen: https://riptutorial.com/de/elasticsearch/topic/10745/aggregationen

Kapitel 3: Analysatoren

Bemerkungen

Analyzer nehmen den Text aus einem String-Feld und generieren Token, die beim Abfragen verwendet werden.

Ein Analyzer arbeitet in einer Reihenfolge:

- `CharFilters` (Null oder mehr)
- `Tokenizer` (Eins)
- `TokenFilters` (Null oder mehr)

Der Analysator kann auf Zuordnungen angewendet werden, sodass Felder bei der Indexierung auf Token-Basis und nicht auf der gesamten Zeichenfolge durchgeführt werden. Bei der Abfrage wird die Eingabezeichenfolge auch durch den Analysator geleitet. Wenn Sie also Text im Analyzer normalisieren, stimmt der Text immer überein, selbst wenn die Abfrage eine nicht normalisierte Zeichenfolge enthält.

Examples

Kartierung

Ein Analyzer kann mithilfe von "Analyzer" auf ein Mapping angewendet werden. Standardmäßig wird der "Standard" Analyzer verwendet. Wenn Sie keinen Analysator verwenden möchten (da die Tokenisierung oder Normalisierung nicht sinnvoll wäre), können Sie alternativ "index" angeben: "not_analyzed".

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

Multi-Felder

Manchmal kann es nützlich sein, mehrere verschiedene Indizes eines Feldes mit

unterschiedlichen Analysatoren zu haben. Sie können dazu die Mehrfeldfunktion verwenden.

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

Bei der Abfrage können Sie anstelle von "user.name" (das in diesem Fall noch den Standard Analyzer verwenden würde) einfach "user.name.special" oder "user.name.unanalyzed" verwenden. Beachten Sie, dass das Dokument unverändert bleibt. Dies wirkt sich nur auf die Indizierung aus.

Analysatoren

Die Analyse in Elasticsearch kommt in einen Kontext, wenn Sie die Daten in Ihrem Index analysieren möchten.

Mit Analysatoren können wir Folgendes ausführen:

- Abkürzungen
- Stemming
- Tippfehler

Wir werden uns jetzt alle ansehen.

1. **Abkürzungen** :

   Mithilfe von Analysatoren können wir elasticsearch sagen, wie Abkürzungen in unseren Daten behandelt werden sollen, z.

2. **Stemming** :

   Die Verwendung von "stemming" in Analysatoren ermöglicht es uns, Basiswörter für geänderte Verben wie zu verwenden

Wort	Änderungen
benötigen	Voraussetzung, erforderlich

3. **Typo Handling** :

   Analysatoren bieten auch Tippfehler an, da bei der Abfrage, ob nach einem bestimmten Wort gesucht wird, sagen wir "Wiederauferstehung". Dann gibt elasticsearch die Ergebnisse zurück, bei denen Tippfehler vorhanden sind. Tippfehler behandeln Tippfehler wie Wiederauferstehung, Wiederbelebung als gleich und führen das Ergebnis erneut aus.

Wort	Änderungen
Auferstehung	Wiederauferstehung, Wiederauferstehung

Analysatoren in der Elasticsearch

1. Standard
2. Einfach
3. Whitespace
4. Halt
5. Stichwort
6. Muster
7. Sprache
8. Schneeball

Case Analyzer ignorieren

Manchmal müssen wir den Fall unserer Abfrage in Bezug auf die Übereinstimmung im Dokument ignorieren. In diesem Fall kann ein Analysator verwendet werden, um den Fall während der Suche zu ignorieren. Jedes Feld muss dieses Analysegerät in seiner Eigenschaft enthalten, damit es funktionieren kann:

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

Analysatoren online lesen: https://riptutorial.com/de/elasticsearch/topic/6232/analysatoren

Kapitel 4: Cluster

Bemerkungen

Die Clusterintegrität enthält viele Informationen zum Cluster, z. B. die Anzahl der zugewiesenen Shards ("aktiv") sowie die Anzahl der nicht zugewiesenen und verlagerten Shards. Außerdem gibt es die aktuelle Anzahl der Knoten und Datenknoten im Cluster an, mit denen Sie nach fehlenden Knoten abfragen können (z. B. wenn Sie erwarten, dass es `15` , aber es zeigt nur `14` , dann fehlt ein Knoten.) .

Für jemanden, der sich mit Elasticsearch auskennt, können "zugewiesene" und "nicht zugewiesene" Shards ihnen dabei helfen, Probleme aufzuspüren.

Das am häufigsten von Cluster Health überprüfte Feld ist der `status` , der sich in einem von drei `status` kann:

- rot
- Gelb
- Grün

Die Farben bedeuten jeweils eine - und nur eine - sehr einfache Sache:

1. Rot bedeutet, dass *mindestens* ein primärer Shard fehlt.
    - Ein fehlendes primäres Shard bedeutet, dass ein Index in den meisten Fällen nicht zum Schreiben (Indexieren) neuer Daten verwendet werden kann.
        - Technisch gesehen können Sie immer noch zu allen primären Shards indexieren, die in diesem Index verfügbar sind. In der Praxis bedeutet dies jedoch, dass Sie dies nicht tun können, weil Sie nicht generell steuern, welche Shards ein bestimmtes Dokument erhalten.
        - Die Suche ist immer noch gegen einen roten Cluster möglich. Dies bedeutet jedoch, dass Sie Teilergebnisse erhalten, wenn einem Index, den Sie durchsuchen, Shards fehlen.
    - Unter normalen Umständen bedeutet dies nur, dass der primäre Shard zugewiesen wird (`initializing_shards`).
    - Wenn ein Knoten den Cluster gerade verlassen hat (z. B. weil die Maschine, an der er ausgeführt wurde, seine Stromversorgung verloren hat), ist es sinnvoll, dass Sie einige primäre Shards *zeitweilig* verpassen.
        - Jeder Replikat-Shard für diesen primären Shard wird in diesem Szenario zum primären Shard.
2. Gelb bedeutet, dass alle primären Shards aktiv sind, aber *mindestens* ein Replikat-Shard fehlt.
    - Eine fehlende Replik wirkt sich nur auf die Indizierung aus, wenn die Konsistenzeinstellungen Auswirkungen auf die Indizierung haben.
        - Standardmäßig gibt es nur ein Replikat für eine Primärdatenbank. Die Indexierung kann bei einem einzelnen fehlenden Replikat erfolgen.
    - Unter normalen Umständen bedeutet dies nur, dass das Replikat-Shard zugewiesen

wird (`initializing_shards`).
- Ein Ein-Knoten-Cluster mit aktivierten Replikaten ist *bestenfalls immer* gelb. Es kann rot sein, wenn noch kein primärer Shard zugewiesen wurde.
    - Wenn Sie nur einen einzelnen Knoten haben, ist es sinnvoll, Replikate zu deaktivieren, da Sie keine erwarten. Dann kann es grün sein.

3. Grün zeigt an, dass alle Shards aktiv sind.
    - Die einzige Shard-Aktivität, die für einen grünen Cluster zulässig ist, ist `relocating_shards` .
    - Neue Indizes und daher auch neue Shards führen dazu, dass der Cluster von Rot zu Gelb zu Grün wechselt, da jeder Shard zugewiesen wird (primärer zuerst, gelb und dann Repliken, wenn möglich, grün).
        - In Elasticsearch 5.x und später werden neue Indizes **nicht** der Cluster rot machen , wenn es ihnen zu lange zuzuteilen nimmt.

Examples

Von Menschen lesbare, tabellarische Cluster-Gesundheit mit Kopfzeilen

Beispiel verwendet grundlegende HTTP-Syntax. Alle `<#>` im Beispiel sollten beim Kopieren entfernt werden.

Sie können die `_cat` APIs verwenden, um aus verschiedenen Gründen eine lesbare tabellarische Ausgabe zu erhalten.

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

1. Das `?v` ist optional, impliziert jedoch, dass Sie eine "verbose" Ausgabe wünschen.

`_cat/health` existiert seit Elasticsearch 1.x, aber hier ist ein Beispiel für die Ausgabe von Elasticsearch 5.x:

Mit ausführlicher Ausgabe:

```
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%
```

Von Menschen lesbarer, tabellarischer Cluster Health ohne Kopfzeilen

Beispiel verwendet grundlegende HTTP-Syntax. Alle `<#>` im Beispiel sollten beim Kopieren entfernt werden.

Sie können die `_cat` APIs verwenden, um aus verschiedenen Gründen eine lesbare tabellarische Ausgabe zu erhalten.

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

`_cat/health` existiert seit Elasticsearch 1.x, aber hier ist ein Beispiel für die Ausgabe von Elasticsearch 5.x:

Ohne ausführliche Ausgabe:

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

Von Menschen lesbarer, tabellarischer Cluster Health mit ausgewählten Kopfzeilen

Beispiel verwendet grundlegende HTTP-Syntax. Alle `<#>` im Beispiel sollten beim Kopieren entfernt werden.

Wie die meisten `_cat` APIs in Elasticsearch reagiert die API selektiv mit einem Standardsatz von Feldern. Es gibt jedoch auch andere Felder aus der API, wenn Sie sie möchten:

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

1. `?help` bewirkt `?help` dass die API die Felder (und Kurznamen) sowie eine kurze Beschreibung zurückgibt.

`_cat/health` existiert seit Elasticsearch 1.x, aber hier ist ein Beispiel für die Ausgabe von Elasticsearch 5.x:

Felder, die ab dem Erstellungsdatum dieses Beispiels verfügbar sind:

```
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
```

Sie können dies dann verwenden, um nur die Felder zu drucken:

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

1. `h=...` definiert die Liste der Felder, die zurückgegeben werden sollen.
2. `v` (ausführlich) definiert, dass die Kopfzeilen gedruckt werden sollen.

Die Ausgabe von einer Instanz von Elasticsearch 5.x:

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

JSON-basierter Clusterzustand

Beispiel verwendet grundlegende HTTP-Syntax. Alle `<#>` im Beispiel sollten beim Kopieren entfernt werden.

Die `_cat` APIs sind häufig für `_cat` , um auf einen Blick Details über den Cluster zu erhalten. Sie möchten jedoch häufig konsistent analysierbare Ausgaben für die Verwendung mit Software. Im Allgemeinen sind die JSON-APIs für diesen Zweck gedacht.

```
GET /_cluster/health
```

`_cluster/health` existiert seit Elasticsearch 1.x, aber hier ist ein Beispiel für die Ausgabe von 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
}
```

Cluster online lesen: https://riptutorial.com/de/elasticsearch/topic/2069/cluster

Kapitel 5: Curl-Befehle

Syntax

- curl -X <VERB> '<PROTOKOLL>: // <HOST>: <PORT> / <PFAD>? <QUERY_STRING>' -d '<BODY>'
- Woher:
- VERB: Die entsprechende HTTP-Methode oder das entsprechende Verb: GET, POST, PUT, HEAD oder DELETE
- PROTOKOLL: Entweder http oder https (wenn Sie einen https-Proxy vor Elasticsearch haben.)
- HOST: Der Hostname eines Knotens in Ihrem Elasticsearch-Cluster oder localhost für einen Knoten auf Ihrem lokalen Computer.
- PORT: Der Port, auf dem der Elasticsearch-HTTP-Dienst ausgeführt wird. Der Standardwert ist 9200.
- PATH: API Endpoint (beispielsweise gibt _count die Anzahl der Dokumente im Cluster zurück). Der Pfad kann mehrere Komponenten enthalten, z. B. _cluster / stats oder _nodes / stats / jvm
- QUERY_STRING: Beliebige optionale Parameter für die Abfragezeichenfolge (zum Beispiel: Die JSON-Antwort wird ziemlich gedruckt, um das Lesen zu erleichtern.)
- KÖRPER: Ein JSON-codierter Anforderungshauptteil (falls für die Anforderung ein solcher erforderlich ist.)
- Referenz: Im Gespräch mit Elasticsearch: Elasticsearch Docs

Examples

Curl Befehl zum Zählen der Anzahl der Dokumente im Cluster

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

Ausgabe:

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

```
  }
}
```

Der Index enthält 90 Dokumente.

Referenzlink: Hier

Dokument nach ID abrufen

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

Ausgabe:

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

Referenzlink: Hier

Erstellen Sie einen Index

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

Ausgabe:

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

Referenzlink: Hier

Alle Indizes auflisten

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

Ausgabe:

```
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
```

Referenzlink: Hier

Einen Index löschen

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

Ausgabe:

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

Referenzlink: Hier

Alle Dokumente in einem Index auflisten

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

Dies verwendet die `Search` API und gibt alle Einträge unter dem Index `myIndexName` .

Referenzlink: Hier

Curl-Befehle online lesen: https://riptutorial.com/de/elasticsearch/topic/3703/curl-befehle

Kapitel 6: Elasticsearch mit Kibana lernen

Einführung

Kibana ist ein Front-End-Datenvisualisierungs-Tool für die Elasticsearch. Informationen zur Installation von Kibana finden Sie in der Dokumentation zu Kibana. Um Kibana auf localhost auszuführen, gehen Sie zu https: // localhost: 5601 und gehen Sie zur kibana-Konsole.

Examples

Erkunde deinen Cluster mit Kibana

Die Befehlssyntax hat folgenden Typ:

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

Führen Sie den folgenden Befehl aus, um das Elasticsearch-Cluster über die Kibana Console zu untersuchen.

- Zur Überprüfung des Clusterzustands

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

- Zur Auflistung aller Indizes

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

- Zur Erstellung eines Indexes mit dem Namen Auto

```
 PUT /car?pretty
```

- Zur Indizierung des Dokuments mit dem Namen auto externen Typs mit der ID 1

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

Die Antwort der obigen Abfrage lautet:

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

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

- das oben genannte Dokument kann mit folgendem Befehl abgerufen werden:

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

- Um einen Index zu löschen

```
DELETE /car?pretty
```

Ändern Sie Ihre Elasticsearch-Daten

Elasticsearch bietet Datenmanipulations- und Datensuchfunktionen in nahezu Echtzeit. In diesem Beispiel haben wir Aktualisierungs-, Lösch- und Stapelverarbeitungsvorgänge.

- Aktualisieren Sie dasselbe Dokument. Angenommen, wir haben bereits ein Dokument unter / car / external / 1 indexiert. Wenn Sie dann den Befehl zum Indizieren der Daten ausführen, wird das vorherige Dokument ersetzt.

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

Das vorherige Fahrzeugdokument mit der ID 1 mit dem Namen "Tata Nexon" wird mit dem neuen Namen "Tata Nexa" aktualisiert.

- Indizierung der Daten mit expliziter ID

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

  Für die Indizierung des Dokuments ohne eine ID verwenden wir das **POST-** Verb anstelle des **PUT-** Verbs. Wenn wir keine ID angeben, generiert elasticsearch eine zufällige ID und verwendet sie dann zur Indizierung des Dokuments.

- Aktualisieren des vorherigen Dokuments mit einer ID teilweise.

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

- Aktualisieren des Dokuments mit zusätzlichen Informationen

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

- Aktualisieren des Dokuments mit einfachen Skripts.

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

ctx._source bezieht sich auf das aktuelle Quelldokument, das gerade aktualisiert wird. Das obige Skript bietet nur ein Skript, das gleichzeitig aktualisiert werden kann.

- Dokument löschen

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

Hinweis: Das Löschen eines gesamten Indexes ist effizienter als das Löschen aller Dokumente mithilfe der Option "Nach Abfrage löschen"

Stapelverarbeitung

Abgesehen von der Indexierung, Aktualisierung und Löschung des Dokuments bietet elasticsearch auch die Möglichkeit, die oben genannten Vorgänge **stapelweise** mithilfe der **_bulk-** API **durchzuführen** .

- zum Aktualisieren mehrerer Dokumente mithilfe der **_bulk-** API

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

- zum Aktualisieren und Löschen der Dokumente mithilfe der **_bulk-** API

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

Wenn eine Operation fehlschlägt, wird die Massen-API nicht angehalten. Es führt alle Vorgänge aus und gibt schließlich den Bericht für alle Vorgänge zurück.

Elasticsearch mit Kibana lernen online lesen:
https://riptutorial.com/de/elasticsearch/topic/10058/elasticsearch-mit-kibana-lernen

Kapitel 7: Elasticsearch-Konfiguration

Bemerkungen

Elasticsearch verfügt über eine Reihe von Standardwerten, die eine gute Erfahrung für die Entwicklung bieten. Die implizite Aussage dort ist, dass es nicht unbedingt großartig für die Produktion ist, die auf Ihre eigenen Bedürfnisse zugeschnitten sein muss und daher nicht vorhergesagt werden kann.

Die Standardeinstellungen erleichtern das Herunterladen und Ausführen mehrerer Knoten *auf derselben Maschine* ohne Konfigurationsänderungen.

Wo sind die einstellungen

In jeder Installation von Elasticsearch befindet sich eine `config/elasticsearch.yml` . Hier leben die folgenden Einstellungen :

- `cluster.name`
  - Der Name des Clusters, dem der Knoten beitritt. Alle Knoten in demselben Cluster **müssen** denselben Namen haben.
  - Derzeit ist die elastische `elasticsearch` standardmäßig `elasticsearch` .
- `node.*`
  - `node.name`
    - Wenn nicht angegeben, wird bei *jedem Start des Knotens* ein zufälliger Name generiert. Das kann Spaß machen, ist aber für Produktionsumgebungen nicht geeignet.
    - Namen müssen *nicht* eindeutig sein, sie **sollten** jedoch eindeutig sein.
  - `node.master`
    - Eine boolesche Einstellung. Wenn `true` , bedeutet dies, dass der Knoten ein in Frage kommender Master-Knoten ist und er kann *der* ausgewählte Master-Knoten sein.
    - Der Standardwert ist " `true` , dh jeder Knoten ist ein in Frage kommender Master-Knoten.
  - `node.data`
    - Eine boolesche Einstellung. Wenn dies `true` , bedeutet dies, dass der Knoten Daten speichert und Suchaktivitäten verarbeitet.
    - Der Standardwert ist " `true` .
- `path.*`
  - `path.data`
    - Der Ort, an dem Dateien für den Knoten geschrieben werden. *Alle Knoten verwenden dieses Verzeichnis* zum Speichern von Metadaten, Datenknoten jedoch auch zum Speichern / Indexieren von Dokumenten.
    - `./data` .
      - Dies bedeutet, dass `data` für Sie als Peer-Verzeichnis für die `config` *innerhalb* des Elasticsearch-Verzeichnisses erstellt werden.
  - `path.logs`
    - Der Speicherort, an den Protokolldateien geschrieben werden.

        - `./logs` .
- `network.*`
    - `network.host`
        - Der `_local_` ist `_local_` , was effektiv `localhost` .
            - Das bedeutet, dass Knoten standardmäßig nicht von außerhalb der aktuellen Maschine kommuniziert werden können!
    - `network.bind_host`
        - Möglicherweise ein Array. Dies teilt Elasticsearch mit, welche Adressen der aktuellen Maschine auch Sockets binden sollen.
            - Mit dieser Liste können Maschinen von außerhalb der Maschine (z. B. andere Knoten im Cluster) mit diesem Knoten kommunizieren.
        - Standardmäßig ist `network.host` .
    - `network.publish_host`
        - Ein einzelner Host, mit dem anderen Knoten mitgeteilt wird, wie sie am besten mit diesem Knoten kommunizieren.
            - Wenn Sie ein Array an `network.bind_host` , sollte dies *derjenige* Host sein, der für die Kommunikation zwischen Knoten verwendet werden soll.
        - Voreinstellung für network.host`.
- `discovery.zen.*`
    - `discovery.zen.minimum_master_nodes`
        - Definiert das Quorum für die Masterwahl. Dies **muss** mithilfe der folgenden Gleichung festgelegt werden: `(M / 2) + 1` wobei `M` die Anzahl der in *Frage* `node.master: true` Master-Knoten ist (Knoten, die `node.master: true` implizit oder explizit).
        - Der Standardwert ist `1` , was nur für einen einzelnen Knotencluster gilt!
    - `discovery.zen.ping.unicast.hosts`
        - Der Mechanismus zum Verbinden dieses Knotens mit dem Rest eines Clusters.
        - Hier *sollten* geeignete Hauptknoten aufgelistet werden, damit ein Knoten den Rest des Clusters finden kann.
        - Der Wert, der hier verwendet werden soll, ist der `network.publish_host` dieser anderen Knoten.
        - Der Standardwert ist `localhost` Dies bedeutet, dass auf dem lokalen Computer nur nach einem Cluster gesucht wird.

Welche Einstellungen gibt es?

Elasticsearch bietet drei verschiedene Arten von Einstellungen:

- Clusterweite Einstellungen
    - Dies sind Einstellungen, die für alles im Cluster gelten, beispielsweise für alle Knoten oder alle Indizes.
- Knoteneinstellungen
    - Diese Einstellungen gelten nur für den aktuellen Knoten.
- Indexeinstellungen

    - Diese Einstellungen gelten nur für den Index.

Je nach Einstellung kann es sein:

- Zur Laufzeit dynamisch geändert
- Geändert nach einem Neustart (Schließen / Öffnen) des Index
    - Einige Einstellungen auf Indexebene erfordern nicht, dass der Index geschlossen und erneut geöffnet wird. Es kann jedoch erforderlich sein, dass der Index zwangsweise erneut zusammengefügt wird, damit die Einstellung wirksam wird.
        - Der Kompressionsgrad eines Index ist ein Beispiel für diesen Einstellungstyp. Sie kann dynamisch geändert werden, aber nur neue *Segmente* nutzen die Änderung. Wenn sich also ein Index nicht ändert, nutzt er die Änderung niemals aus, es sei denn, Sie erzwingen, dass der Index seine Segmente neu erstellt.
- Geändert nach einem Neustart des Knotens
- Geändert nach einem Neustart des Clusters
- Nie geändert

Überprüfen Sie immer die Dokumentation Ihrer Version von Elasticsearch, was Sie mit einer Einstellung tun können oder nicht.

Wie kann ich Einstellungen vornehmen?

Sie können einige Einstellungen festlegen, von denen einige nicht vorgeschlagen werden:

- Kommandozeilenargumente

In Elasticsearch 1.x und 2.x können Sie die meisten Einstellungen als Java-Systemeigenschaften mit dem Präfix `es.` :

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

In Elasticsearch 5.x ändert sich dies, um die Verwendung von Java-Systemeigenschaften zu vermeiden, statt eines benutzerdefinierten Argumenttyps mit `-E` anstelle von `-Des.` :

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

Diese Methode zum Anwenden von Einstellungen eignet sich hervorragend, wenn Sie zum Starten und Stoppen des Clusters Tools wie Puppet, Chef oder Ansible verwenden. Es funktioniert jedoch sehr schlecht, wenn es manuell ausgeführt wird.

- YAML-Einstellungen
    - In Beispielen gezeigt
- Dynamische Einstellungen
    - In Beispielen gezeigt

Die Reihenfolge, in der die Einstellungen angewendet werden, liegt in der Reihenfolge der dynamischsten:

1. Transiente Einstellungen
2. Ständige Einstellungen
3. Befehlszeileneinstellungen
4. YAML (statische) Einstellungen

Wenn die Einstellung zweimal auf einer dieser Ebenen eingestellt ist, wird die höchste Stufe wirksam.

Examples

Statische Elasticsearch-Einstellungen

Elasticsearch verwendet eine YAML-Konfigurationsdatei (Yet Another Another Markup Language), die sich im Standard-Elasticsearch-Verzeichnis befindet (RPM- und DEB-Installationen ändern diesen Speicherort unter anderem).

Sie können grundlegende Einstellungen in `config/elasticsearch.yml` vornehmen:

```
# 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_
```

Permanente dynamische Clustereinstellungen

Wenn Sie eine Einstellung dynamisch anwenden müssen, nachdem der Cluster bereits gestartet wurde und tatsächlich dynamisch festgelegt werden kann, können Sie sie mit der `_cluster/settings` API `_cluster/settings` .

Persistente Einstellungen sind eine der zwei Arten von clusterweiten Einstellungen, die angewendet werden können. Eine hartnäckige Einstellung **wird** einen vollständigen Cluster - Neustart überleben.

Hinweis: Nicht alle Einstellungen können dynamisch angewendet werden. Beispielsweise kann der Name des Clusters nicht dynamisch umbenannt werden. Die meisten Einstellungen auf Knotenebene können auch nicht dynamisch festgelegt werden (da sie nicht einzeln anvisiert werden können).

Dies ist **nicht** die API, die zum Einstellen der Einstellungen auf Indexebene verwendet wird. Sie können erkennen, dass es sich bei der Einstellung um eine Indexebene handelt, da sie mit dem `index.` . Einstellungen, deren Name in Form von `indices.` vorliegt `indices.` *sind* clusterweite Einstellungen, da sie für alle Indizes gelten.

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

Warnung : In Elasticsearch 1.x und 2.x können Sie eine permanente Einstellung nicht *aufheben* .

Glücklicherweise wurde dies in Elasticsearch 5.x verbessert und Sie können eine Einstellung jetzt entfernen, indem Sie sie auf `null` :

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

Eine nicht festgelegte Einstellung wird auf den Standardwert oder einen Wert zurückgesetzt, der auf einer niedrigeren Prioritätsstufe definiert ist (z. B. Befehlszeileneinstellungen).

Einstellungen für transiente dynamische Cluster

Wenn Sie eine Einstellung dynamisch anwenden müssen, nachdem der Cluster bereits gestartet wurde und tatsächlich dynamisch festgelegt werden kann, können Sie sie mit der `_cluster/settings` API `_cluster/settings` .

Transiente Einstellungen sind eine der zwei Arten von clusterweiten Einstellungen, die angewendet werden können. Eine vorübergehende Einstellung wird **keinen** vollständigen Cluster - Neustart überleben.

Hinweis: Nicht alle Einstellungen können dynamisch angewendet werden. Beispielsweise kann der Name des Clusters nicht dynamisch umbenannt werden. Die meisten Einstellungen auf Knotenebene können auch nicht dynamisch festgelegt werden (da sie nicht einzeln anvisiert werden können).

Dies ist **nicht** die API, die zum Einstellen der Einstellungen auf Indexebene verwendet wird. Sie können erkennen, dass es sich bei der Einstellung um eine Indexebene handelt, da sie mit dem `index.` . Einstellungen, deren Name in Form von `indices.` vorliegt `indices.` *sind* clusterweite Einstellungen, da sie für alle Indizes gelten.

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

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

Warnung : In Elasticsearch 1.x und 2.x können Sie keine Übergangseinstellungen ohne einen vollständigen Neustart des Clusters aufheben.

Glücklicherweise wurde dies in Elasticsearch 5.x verbessert und Sie können eine Einstellung jetzt entfernen, indem Sie sie auf null setzen:

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

Eine nicht festgelegte Einstellung wird auf den Standardwert oder einen Wert zurückgesetzt, der auf einer niedrigeren Prioritätsstufe definiert ist (z. B. `persistent` Einstellungen).

Indexeinstellungen

Indexeinstellungen sind Einstellungen, die für einen einzelnen Index gelten. Solche Einstellungen beginnen mit dem `index.` . Die Ausnahme von dieser Regel sind `number_of_shards` und `number_of_replicas` , die auch in der Form `index.number_of_shards` und `index.number_of_replicas` .

Wie der Name vermuten lässt, gelten die Einstellungen auf Indexebene für einen einzelnen Index. Einige Einstellungen müssen zum Zeitpunkt der Erstellung angewendet werden, da sie nicht dynamisch geändert werden können, z. B. die Einstellung `index.number_of_shards` , die die Anzahl der primären Shards für den Index steuert.

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

oder, in einem übersichtlicheren Format, können Sie die Schlüsselpräfixe jeweils miteinander kombinieren `.` :

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

Die obigen Beispiele erstellen einen Index mit den angegebenen Einstellungen. Sie können die Einstellungen pro Index dynamisch ändern, indem Sie den Endpunkt des Index `_settings` . Zum Beispiel ändern wir hier die Slowlog-Einstellungen *nur* für die Warnstufe :

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

Warnung : Elasticsearch 1.x und 2.x haben die Einstellungsnamen auf Indexebene nicht sehr streng überprüft. Wenn Sie einen Tippfehler hätten oder einfach eine Einstellung vorgenommen hätten, würde er diese blind akzeptieren, ansonsten aber ignorieren. Elasticsearch 5.x überprüft die Namen der Einstellungen streng und lehnt jeden Versuch ab, Indexeinstellungen mit unbekannten Einstellungen anzuwenden (aufgrund von Tippfehlern oder fehlendem Plug-In). Beide Anweisungen gelten für dynamisch sich ändernde Indexeinstellungen und zum Zeitpunkt der Erstellung.

Dynamische Indexeinstellungen für mehrere Indizes gleichzeitig

Sie können die gleiche Änderung in der gezeigten Anwendung `Index Settings` - `Index Settings` Beispiel für *alle* vorhandenen Indizes mit einer Anfrage oder sogar eine Teilmenge von ihnen:

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

oder

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

oder

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

Wenn Sie es vorziehen, es auch selektiver zu machen, können Sie mehrere auswählen, ohne alle zu liefern:

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

Elasticsearch-Konfiguration online lesen:
https://riptutorial.com/de/elasticsearch/topic/3411/elasticsearch-konfiguration

Kapitel 8: Python-Schnittstelle

Parameter

Parameter	Einzelheiten
Gastgeber	Array von Hosts in Form eines Objekts, das die Schlüssel `host` und `port` . Standard - `host` ist 'localhost' und `port` ist 9200. Ein Beispiel - Eintrag sieht aus wie `[{"host": "ip of es server", "port": 9200}]` - `[{"host": "ip of es server", "port": 9200}]`
sniff_on_start	Boolean Wenn der Client beim Start Knoten ausloten soll, bedeutet "Sniffing" eine Liste der Knoten im Elasticsearch-Cluster
sniff_on_connection_fail	Boolean zum Auslösen von Sniffing, wenn die Verbindung bei aktivem Client fehlschlägt
sniffer_timeout	Zeitunterschied in Sekunden zwischen jedem Sniff
sniff_timeout	Zeit für eine einzige Aufforderung zum Schnüffeln in Sekunden
retry_on_timeout	Booelan für den Fall, dass der Client ein Timeout auslösen sollte, wenn er einen anderen Elasticsearch-Knoten kontaktiert, oder einfach einen Fehler auslöst
http_auth	Die http-Basisauthentifizierung kann hier in Form von `username:password` bereitgestellt werden

Examples

Dokumentieren eines Dokuments (z. B. Hinzufügen einer Probe)

Installieren Sie die erforderliche Python-Bibliothek über:

```
$ pip install elasticsearch
```

Stellen Sie eine Verbindung zu Elasticsearch her, erstellen Sie ein Dokument (z. B. Dateneingabe) und "indexieren" Sie das Dokument mithilfe von 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"])
```

Verbindung zu einem Cluster

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

Einen leeren Index erstellen und das Mapping festlegen

In diesem Beispiel erstellen wir einen leeren Index (wir indizieren keine Dokumente darin), indem wir seine Zuordnung definieren.

Zuerst erstellen wir eine `ElasticSearch` Instanz und definieren dann das Mapping unserer Wahl. Als Nächstes prüfen wir, ob der Index vorhanden ist. Wenn dies nicht der Fall ist, erstellen Sie ihn, indem Sie die `index` und `body` Parameter angeben, die den Indexnamen bzw. den Body der Zuordnung enthalten.

```
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))
```

Partielle Aktualisierung und Aktualisierung per Abfrage

Partial Update: Wird verwendet , wenn ein Teildokumenten Aktualisierung erforderlich ist, dh in dem folgenden Beispiel das Feld zu tun , `name` des Dokuments mit der ID `doc_id` zu ‚John' aktualisiert werden soll. Wenn das Feld fehlt, wird es einfach zum Dokument hinzugefügt.

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

Update durch Abfrage: Wird verwendet , wenn benötigt wird , um Dokumente zu aktualisieren , die einen Zustand, dh im folgenden Beispiel erfüllen aktualisieren wir das Alter der Dokumente , deren `name` Feld Streichhölzer ‚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')
```

Python-Schnittstelle online lesen: https://riptutorial.com/de/elasticsearch/topic/2068/python-schnittstelle

Kapitel 9: Such-API

Einführung

Mit der Such-API können Sie eine Suchabfrage ausführen und Suchtreffer abrufen, die der Abfrage entsprechen. Die Abfrage kann entweder mit einer einfachen Abfragezeichenfolge als Parameter oder mit einem Anfragetext bereitgestellt werden.

Examples

Routing

Wenn Sie eine Suche ausführen, wird sie an alle Index- / Index-Shards gesendet (Round-Robin zwischen Replikaten). Welche Shards gesucht werden, kann durch Angabe des Routing-Parameters gesteuert werden. Wenn Sie zum Beispiel Tweets indizieren, kann der Routing-Wert der Benutzername sein:

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

Suche mithilfe des Anfragetextes

Die Suche kann auch über Elasticsearch mit einem Such-DSL durchgeführt werden. Das Abfrageelement innerhalb des Suchanforderungstextes ermöglicht das Definieren einer Abfrage mit dem Query DSL.

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

Multi-Suche

Mit der Option multi_search können Sie eine Abfrage in mehreren Feldern gleichzeitig suchen.

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

```
  }
}
```

Wir können auch die Punktzahl bestimmter Felder mit dem Boost-Operator (^) erhöhen und Platzhalter im Feldnamen (*) verwenden.

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

URI-Suche und Hervorhebung

Eine Suchabfrage kann ausschließlich unter Verwendung eines URI ausgeführt werden, indem Anforderungsparameter angegeben werden. Nicht alle Suchoptionen werden angezeigt, wenn eine Suche mit diesem Modus ausgeführt wird. Dies kann jedoch für schnelle "Curl-Tests" nützlich sein.

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

Eine weitere nützliche Funktion ist das Hervorheben der Treffer in den Dokumenten.

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

In diesem Fall wird das jeweilige Feld für jeden Suchtreffer hervorgehoben

Such-API online lesen: https://riptutorial.com/de/elasticsearch/topic/8625/such-api

Kapitel 10: Unterschied zwischen Indizes und Typen

Bemerkungen

Es ist leicht, `type` wie eine Tabelle in einer SQL-Datenbank zu sehen, wobei der `index` die SQL-Datenbank ist. Dies ist jedoch keine gute Möglichkeit, sich dem `type` s zu nähern.

Alles über Typen

Tatsächlich sind Typen *buchstäblich* nur ein Metadatenfeld, das jedem Dokument von Elasticsearch hinzugefügt wird: `_type` . In den obigen Beispielen wurden zwei Typen erstellt: `my_type` und `my_other_type` . Das bedeutet, dass jedes Dokument, das den Typen zugeordnet ist, ein zusätzliches Feld hat, das automatisch wie `"_type": "my_type"` ; Dieses Dokument wird mit dem Dokument indiziert, wodurch es zu einem *durchsuchbaren oder filterbaren Feld* wird. Es hat jedoch keine Auswirkungen auf das Rohdokument selbst, sodass sich Ihre Anwendung nicht darum kümmern muss.

Alle Typen leben in demselben Index und daher in denselben gemeinsamen Shards des Index. Selbst auf der Festplattenebene leben sie in den gleichen Dateien. Die einzige Trennung, die das Erstellen eines zweiten Typs bietet, ist eine logische. Jeder Typ, unabhängig davon, ob er eindeutig ist oder nicht, muss in den Mappings vorhanden sein, und alle diese Mappings müssen in Ihrem Clusterstatus vorhanden sein. Dies frisst Speicher und wenn jeder Typ dynamisch aktualisiert wird, nimmt die Leistung ab, wenn sich die Zuordnungen ändern.

Daher empfiehlt es sich, nur einen einzigen Typ zu definieren, es sei denn, Sie benötigen tatsächlich andere Typen. Es werden häufig Szenarien angezeigt, in denen mehrere Typen wünschenswert sind. Stellen Sie sich beispielsweise vor, Sie hätten einen Autoindex. Es kann für Sie nützlich sein, es mit mehreren Typen aufzuschlüsseln:

- BMW
- chevy
- Honda
- Mazda
- Mercedes
- Nissan
- Rangerover
- Toyota
- ...

Auf diese Weise können Sie nach allen Fahrzeugen suchen oder sie nach Hersteller einschränken. Der Unterschied zwischen diesen beiden Suchen ist so einfach:

```
GET /cars/_search
```

und

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

Was für neue Benutzer von Elasticsearch nicht offensichtlich ist, ist, dass die zweite Form eine Spezialisierung der ersten Form ist. Es wird buchstäblich umgeschrieben in:

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

Es filtert einfach alle Dokumente heraus, die nicht mit einem `_type` Feld mit dem Wert `bmw` . Da jedes Dokument mit seinem Typ als `_type` Feld indiziert wird, dient dies als ziemlich einfacher Filter. Wenn in einem der Beispiele eine tatsächliche Suche bereitgestellt wurde, wird der Filter entsprechend der vollständigen Suche hinzugefügt.

Wenn also die Typen identisch sind, ist es viel besser, einen einzigen Typ (z. B. `manufacturer` in diesem Beispiel) bereitzustellen und ihn effektiv zu ignorieren. Geben Sie dann in jedem Dokument explizit ein Feld mit dem Namen `make` *oder einen beliebigen Namen an* und filtern *Sie es* manuell, wann immer Sie es einschränken möchten. Dadurch wird die Größe Ihrer Zuordnungen auf `1/n` reduziert, wobei `n` die Anzahl der separaten Typen ist. Es fügt jedem Dokument ein weiteres Feld hinzu, da sonst die Zuordnung vereinfacht wird.

In Elasticsearch 1.x und 2.x sollte ein solches Feld als definiert werden

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

1. Der Name ist beliebig.
2. Der Name ist beliebig *und* könnte mit dem Typnamen übereinstimmen, wenn Sie es auch wollten.

In Elasticsearch 5.x funktioniert das oben immer noch (es ist veraltet), aber der bessere Weg ist

zu verwenden:

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

1. Der Name ist beliebig.
2. Der Name ist beliebig *und* könnte mit dem Typnamen übereinstimmen, wenn Sie es auch wollten.

Typen sollten in Ihren Indizes sparsam verwendet werden, da sie die Indexzuordnungen aufblähen, normalerweise ohne großen Nutzen. Sie müssen mindestens eine haben, aber es gibt nichts, was besagt, dass Sie mehr als eine haben müssen.

Häufige Fragen

- Was ist, wenn ich zwei (oder mehr) Typen habe, die größtenteils identisch sind, jedoch einige eindeutige Felder pro Typ haben?

Auf der Indexebene gibt es keinen Unterschied zwischen einem Typ, der mit wenigen Feldern verwendet wird, die spärlich verwendet werden, *und* zwischen mehreren Typen, die eine Reihe von nicht-spärlichen Feldern mit einigen nicht gemeinsam genutzten Feldern verwenden (dh der andere Typ verwendet das Feld niemals selbst) (s)).

Anders gesagt: Ein spärlich genutztes Feld ist *unabhängig vom Typ* im Index spärlich. Die Sparsity ist für den Index nicht von Vorteil oder schadet ihm nicht, nur weil er in einem separaten Typ definiert ist.

Sie sollten diese Typen nur kombinieren und ein separates Typenfeld hinzufügen.

- Warum müssen separate Typen Felder genau gleich definieren?

Denn jedes Feld wird auf Lucene-Ebene wirklich nur einmal definiert, unabhängig davon, wie viele Typen vorhanden sind. Die Tatsache, dass Typen überhaupt existieren, ist ein Merkmal von Elasticsearch und ist *nur* eine logische Trennung.

- Kann ich separate Typen definieren, bei denen das gleiche Feld anders definiert ist?

Wenn Sie in ES 2.x oder höher einen Weg finden, sollten Sie einen Fehlerbericht öffnen . Wie bereits in der vorigen Frage erwähnt, betrachtet Lucene sie alle als ein einziges Feld, so dass es keine Möglichkeit gibt, diese Arbeit angemessen zu gestalten.

ES 1.x beließ dies als eine implizite Anforderung, die es Benutzern ermöglichte, Bedingungen zu

erstellen, bei denen die Zuordnungen eines Shards in einem Index tatsächlich von einem anderen Shard im selben Index abweichen. Dies war effektiv eine Wettlaufsituation und *konnte* zu unerwarteten Problemen führen.

Ausnahmen von der Regel

- Übergeordnete / untergeordnete Dokumente **erfordern die Verwendung** separater Typen innerhalb desselben Index.
    - Der Elternteil lebt in einem Typ.
    - Das Kind lebt in einem separaten Typ (aber jedes Kind lebt in demselben *Shard* wie sein Elternteil).
- Extrem Nischenanwendungen, bei denen die Erstellung von Tonnen von Indizes unerwünscht ist und die Auswirkung spärlicher Felder der Alternative vorzuziehen ist.
    - Das Elasticsearch Monitoring-Plugin Marvel (1.x und 2.x) oder X-Pack Monitoring (5.x +) überwacht Elasticsearch selbst auf Änderungen im Cluster, Knoten, Indizes, bestimmte Indizes (Indexebene), und sogar Scherben. Es könnten täglich mehr als 5 Indizes erstellt werden, um die Dokumente mit eindeutigen Zuordnungen zu isolieren, *oder* es kann gegen bewährte Methoden verstoßen, um die Clusterlast durch die gemeinsame Nutzung eines Index zu reduzieren (Hinweis: Die Anzahl der definierten Zuordnungen ist praktisch die gleiche, jedoch die Anzahl der erstellten Indizes wird von `n` auf 1) reduziert.
    - Dies ist ein erweitertes Szenario, aber Sie müssen die gemeinsam genutzten Felddefinitionen über Typen hinweg berücksichtigen!

Examples

Einen Index explizit mit einem Typ erstellen

Beispiel verwendet grundlegende HTTP-Funktionen, die leicht in cURL und andere HTTP-Anwendungen übersetzt werden können. Sie stimmen auch mit der Sense- Syntax überein, die in Kibana 5.0 in Console umbenannt wird.

Hinweis: Das Beispiel fügt `<#>` , um die Aufmerksamkeit auf Teile zu lenken. Die sollten entfernt werden, wenn Sie es kopieren!

```
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. Dies erstellt den `index` Verwendung des Endpunkts zum Erstellen des Index.
2. Dadurch wird der `type` .
3. Freigegebene Felder in `type` innerhalb desselben `index` **müssen** dieselbe Definition haben! ES 1.x hat dieses Verhalten nicht strikt durchgesetzt, es war jedoch eine implizite Anforderung. ES 2.x und höher setzen dieses Verhalten strikt durch.
4. Eindeutige Felder über `type` sind in Ordnung.

Indizes (oder Indizes) *enthalten* Typen. Typen sind ein praktischer Mechanismus zum Trennen von Dokumenten. Sie müssen jedoch entweder dynamisch / automatisch oder explizit ein Mapping für jeden verwendeten Typ definieren. Wenn Sie 15 Typen in einem Index definieren, haben Sie 15 eindeutige Zuordnungen.

In den Anmerkungen finden Sie weitere Details zu diesem Konzept und warum Sie Typen verwenden möchten oder nicht.

Dynamisches Erstellen eines Index mit einem Typ

Beispiel verwendet grundlegende HTTP-Funktionen, die leicht in cURL und andere HTTP-Anwendungen übersetzt werden können. Sie stimmen auch mit der Sense- Syntax überein, die in Kibana 5.0 in Console umbenannt wird.

Hinweis: Das Beispiel fügt `<#>` , um die Aufmerksamkeit auf Teile zu lenken. Die sollten entfernt werden, wenn Sie es kopieren!

```
DELETE /my_index <1>

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

1. Falls es bereits existiert (aufgrund eines früheren Beispiels), löschen Sie den Index.
2. Indizieren `my_index` ein Dokument in den Index, `my_index` , mit dem Typ, `my_type` und der ID `abc123` (kann numerisch sein, ist jedoch immer eine Zeichenfolge).
    - Die dynamische Indexerstellung wird standardmäßig durch einfaches Indizieren eines Dokuments aktiviert. Dies ist ideal für Entwicklungsumgebungen, aber nicht unbedingt für Produktionsumgebungen.
3. Dieses Feld ist eine ganze Zahl. Wenn es zum ersten Mal angezeigt wird, muss es zugeordnet werden. Elasticsearch nimmt für jeden eingehenden Typ immer den *breitesten* Typ an, daher würde dieser als `long` statt als `integer` oder `short` abgebildet werden (beide können `1234` und `456`).
4. Dasselbe gilt auch für dieses Feld. Es wird als `double` nicht als `float` abgebildet, wie Sie möchten.

Dieser dynamisch erstellte Index und Typ entspricht in etwa dem im ersten Beispiel definierten Mapping. Es ist jedoch wichtig zu verstehen, wie sich `<3>` und `<4>` auf die automatisch definierten Zuordnungen auswirken.

Sie können dem folgen, indem Sie dynamisch einen anderen Typ zu demselben Index hinzufügen:

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

1. Der Typ ist der einzige Unterschied zum obigen Dokument. Die ID ist die gleiche und das ist in Ordnung! Es hat keine Beziehung zu dem anderen `abc123` außer dass es *zufällig* im selben Index liegt.
2. `field1` bereits im Index vorhanden ist, *muss* es *sich* um denselben `field1` , der in den anderen Typen definiert ist. Das Senden eines Werts, der eine Zeichenfolge oder keine Ganzzahl ist, `"field1": "this is some text"` fehl (z. B. `"field1": "this is some text"` oder `"field1": 123.0`).

Dadurch würden dynamisch die Zuordnungen für `my_other_type` innerhalb desselben Index `my_index` .

Hinweis: Es ist *immer* schneller, Zuordnungen vorab zu definieren, anstatt sie von Elasticsearch zur Indexzeit dynamisch ausführen zu lassen.

Das Endergebnis der Indizierung beider Dokumente wäre dem ersten Beispiel ähnlich, die Feldtypen wären jedoch unterschiedlich und daher etwas verschwenderisch:

```
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. Hierbei wird der Endpunkt `_mappings` , um die Zuordnungen aus dem von uns erstellten Index `_mappings` .
2. Im ersten Schritt dieses Beispiels haben wir `my_type` dynamisch erstellt.
3. `field2` ist jetzt ein `long` statt einer `integer` da wir es nicht im Voraus definiert haben. Dies *kann* sich als verschwenderisch beim Festplattenspeicher erweisen.
4. `object1.field1` ist jetzt aus dem gleichen Grund wie # 3 mit den gleichen Auswirkungen wie # 3 ein `double` .
    - Technisch kann ein `long` in vielen Fällen komprimiert werden. Ein `double` kann jedoch nicht komprimiert werden, da es sich um eine Fließkommazahl handelt.
5. Im zweiten Schritt dieses Beispiels haben wir auch `my_other_type` dynamisch erstellt. Die Abbildung ist zufällig gleich, weil wir bereits `long` und `double` .
    - Denken `field1` daran, dass `field1` mit der Definition von `my_type` `field1` *muss* (und dies auch tut).
    - `field3` ist für diesen Typ einzigartig, daher gibt es keine solche Einschränkung.

Unterschied zwischen Indizes und Typen online lesen:
https://riptutorial.com/de/elasticsearch/topic/3412/unterschied-zwischen-indizes-und-typen

Kapitel 11: Unterschied zwischen relationalen Datenbanken und Elasticsearch

Einführung

Dies ist für die Leser, die aus einem relationalen Hintergrund kommen und Elasticsearch lernen möchten. In diesem Thema werden die Anwendungsfälle beschrieben, für die relationale Datenbanken keine geeignete Option sind.

Examples

Unterschied der Terminologie

Relationale Datenbank	Elasticsearch
Datenbank	Index
Tabelle	Art
Zeile / Aufnahme	Dokumentieren
Spaltenname	Feld

Die obige Tabelle zeigt grob eine Analogie zwischen grundlegenden Elementen der relationalen Datenbank und der Elasticsearch.

Konfiguration

Betrachten der folgenden Struktur in einer relationalen Datenbank:

```
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 Äquivalent:

```
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"
        }
     }
  ]
```

Usecases, bei denen relationale Datenbanken nicht geeignet sind

- Die Essenz der Suche liegt in ihrer Reihenfolge. Jeder möchte, dass Suchergebnisse so angezeigt werden, dass die besten Ergebnisse oben angezeigt werden. Relationale Datenbanken verfügen nicht über eine solche Fähigkeit. Elasticsearch hingegen zeigt Ergebnisse auf der Grundlage der Standardrelevanz.

  Konfiguration

  Gleich wie im vorherigen Beispiel.

  Problemstellung

  Angenommen, der Benutzer möchte nach `shirts` suchen, interessiert sich jedoch für `red` Hemden. In diesem Fall sollten die Ergebnisse mit den Suchbegriffen " `red` und " `shirts` an erster Stelle stehen. Danach sollten Ergebnisse für andere Shirts angezeigt werden.

Lösung mit relationaler Datenbankabfrage

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

Ausgabe

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

Elasticsearch Lösung

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

Ausgabe

```
"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"
     }
  }
 ]
```

Fazit

Wie wir oben sehen können, hat Relational Database Ergebnisse in einer zufälligen Reihenfolge zurückgegeben, während Elasticsearch Ergebnisse in abnehmender Reihenfolge von `_score` die auf der Grundlage der Relevanz berechnet wird.

- Wir neigen dazu, Fehler bei der Eingabe von Suchstrings zu machen. Es gibt Fälle, in denen

der Benutzer einen falschen Suchparameter eingibt. Relationale Datenbanken behandeln solche Fälle nicht. Gummiband zur Rettung.

Konfiguration

Gleich wie im vorherigen Beispiel.

Problemstellung

Angenommen, der Benutzer möchte nach `shirts` suchen, gibt jedoch versehentlich ein falsches Wort `shrt` . Der Benutzer erwartet immer noch die Ergebnisse des Shirts .

Lösung mit relationaler Datenbankabfrage

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

Ausgabe

```
No results found
```

Elasticsearch Lösung

```
POST /test/product/_search

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

Ausgabe

```
 "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"
    }
  }
]
```

Fazit

Wie wir oben sehen können, hat die relationale Datenbank keine Ergebnisse für ein falsches gesuchtes Wort zurückgegeben, während Elasticsearch mit seiner speziellen `fuzzy` Abfrage Ergebnisse zurückgibt.

Unterschied zwischen relationalen Datenbanken und Elasticsearch online lesen: https://riptutorial.com/de/elasticsearch/topic/10632/unterschied-zwischen-relationalen-datenbanken-und-elasticsearch

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit 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	Aggregationen	Sid1199
3	Analysatoren	Bhushan Gadekar, Sid1199, Thomas
4	Cluster	Gerardo Rochín, pickypg
5	Curl-Befehle	Fawix, Mrunal Pagnis, Mrunal Pagnis
6	Elasticsearch mit Kibana lernen	sarvajeetsuman
7	Elasticsearch-Konfiguration	pickypg
8	Python-Schnittstelle	aidan.plenert.macdonald, christinabo, KartikKannapur, pickypg, Sumit Kumar
9	Such-API	aerokite, Sid1199
10	Unterschied zwischen Indizes und Typen	pickypg
11	Unterschied zwischen relationalen Datenbanken und Elasticsearch	Richa