Бесплатная электронная книга

УЧУСЬ

Elasticsearch

Free unaffiliated eBook created from
Stack Overflow contributors.

#elasticsear

ch

Около

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

глава 1: Начало работы с Elasticsearch

замечания

Elasticsearch - это продвинутый сервер поиска с открытым исходным кодом на основе Lucene и написанный на Java.

Он предоставляет распределенные полнофункциональные и частично текстовые, поисковые и геолокационные функции поиска, доступные через HTTP REST API.

Версии

Версия	Дата выхода
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

Версия	Дата выхода
1.2.0	2014-05-22
1.1.0	2014-03-25
1.0.0	2014-02-14

Examples

Установка Elasticsearch на Ubuntu 14.04

Предпосылки

Для запуска Elasticsearch на компьютере требуется Java Runtime Environment (JRE). Elasticsearch требует Java 7 или выше и рекомендует `Oracle JDK version 1.8.0_73` .

Установка 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
```

Проверить версию Java

```
java -version
```

Загрузка и установка пакета

Использование бинарников

1. Загрузите последнюю стабильную версию Elasticsearch здесь .
2. Разархивируйте файл & Run

Linux:

```
$ bin/elasticsearch
```

Использование apt-get

Альтернативой загрузке elasticsearch с веб-сайта является установка его, используя `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
```

Установка elasticsearch версии 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
```

Выполнение службы в Linux:

После установки выше не запускается сама. поэтому нам нужно запустить его как услугу. Как запустить или остановить поиск Elasticsearch зависит от того, использует ли ваша система SysV init или systemd. вы можете проверить его с помощью следующей команды.

```
ps -p 1
```

Если ваш дистрибутив использует SysV init, вам нужно будет запустить:

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

В противном случае, если ваш дистрибутив использует systemd:

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

Запустите команду `CURL` из вашего браузера или клиента REST, чтобы проверить правильность установки Elasticsearch.

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

Установка Elasticsearch в Windows

Предпосылки

Версия Elicsearch для Windows можно получить по этой ссылке: https://www.elastic.co/downloads/elasticsearch . Последний стабильный релиз всегда наверху.

Когда мы устанавливаем в Windows, нам нужен архив `.ZIP` . Нажмите ссылку в разделе « `Downloads:` » и сохраните файл на своем компьютере.

Эта версия эластичной «портативной», то есть вам не нужно запускать установщик для использования программы. Распакуйте содержимое файла в удобное для вас место. Для демонстрации мы предположим, что вы распаковали все на `C:\elasticsearch` .

Обратите внимание, что по умолчанию архив содержит папку с именем `elasticsearch-<version>` , вы можете извлечь эту папку в `C:\` и переименовать ее в `elasticsearch` или создать `C:\elasticsearch` самостоятельно, а затем распаковать только *содержимое* папки в архиве туда.

Поскольку Elasticsearch написан на Java, для его работы требуется среда выполнения Java. Поэтому перед запуском сервера проверьте, доступна ли Java, открыв командную строку и набрав:

```
java -version
```

Вы должны получить ответ, который выглядит так:

```
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)
```

Если вы видите следующее:

> «java» не распознается как внутренняя или внешняя команда, операционная программа или командный файл.

Java не установлен в вашей системе или не настроен должным образом. Вы можете следовать этому руководству, чтобы (повторно) установить Java. Кроме того, убедитесь, что для этих переменных окружения заданы одинаковые значения:

переменная	Значение
JAVA_HOME	C: \ Program Files \ Java \ jre
ДОРОЖКА	...; C: \ Program Files \ Java \ jre

Если вы еще не знаете, как проверить эти переменные, ознакомьтесь с этим руководством .

Запуск из командного файла

С установленной Java откройте папку `bin` . Его можно найти непосредственно в папке, в

которую вы разархивировали все, поэтому она должна находиться в `c:\elasticsearch\bin` . Внутри этой папки находится файл `elasticsearch.bat` который можно использовать для запуска Elasticsearch в окне команд. Это означает, что информация, регистрируемая процессом, будет видна в окне командной строки. Чтобы остановить сервер, нажмите `CTRL C` или просто закройте окно.

Запуск в качестве службы Windows

В идеале вы не хотите иметь дополнительное окно, от которого вы не сможете избавиться во время разработки, и по этой причине Elasticsearch может быть настроен для работы в качестве службы.

Прежде чем мы смогли установить Elasticsearch в качестве сервиса, нам нужно добавить строку в файл `C:\elasticsearch\config\jvm.options` :

> Установщик служб требует, чтобы настройки размера стека потоков были настроены в `jvm.options` перед установкой службы. В 32-битной Windows вы должны добавить `-Xss320k` [...], а в 64-битной Windows вы должны добавить `-Xss1m` в файл `jvm.options` . [источник]

После того как вы сделали это изменение, откройте командную строку и перейдите в каталог `bin` , выполнив следующую команду:

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

Управление услугами обрабатывается `elasticsearch-service.bat` . В старых версиях этот файл можно просто назвать `service.bat` . Чтобы просмотреть все доступные аргументы, запустите его без каких-либо:

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

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

Результат также говорит нам, что есть необязательный аргумент `SERVICE_ID` , но мы можем его игнорировать пока. Чтобы установить службу, просто запустите:

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

После установки службы вы можете запустить и остановить ее с помощью соответствующих аргументов. Чтобы запустить службу, запустите

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

и остановить его, запустить

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

Если вы предпочитаете использовать графический интерфейс для управления службой, вы можете использовать следующую команду:

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

Это откроет Elastic Service Manager, который позволит вам настроить некоторые параметры, связанные с сервисом, а также остановить / запустить службу с помощью кнопок, расположенных в нижней части первой вкладки.

Индексирование и извлечение документа

Доступ к Elasticsearch осуществляется через HTTP REST API, обычно используя библиотеку cURL. Сообщения между сервером поиска и клиентом (ваше или ваше приложение) отправляются в виде строк JSON. По умолчанию Elasticsearch работает на порту 9200.

В приведенных ниже примерах добавлена `?pretty` добавка, чтобы сообщить Elasticsearch о том, чтобы отменить ответ JSON. При использовании этих конечных точек в приложении вам не нужно добавлять этот параметр запроса.

Индексирующие документы

Если мы намерены обновлять информацию в индексе позже, рекомендуется назначить уникальные идентификаторы индексируемым документам. Чтобы добавить документ в индекс с именем `megacorp` , при запуске типа `employee` и 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" ]
}'
```

Отклик:

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "1",
  "_version": 1,
  "_shards": {
    "total": 2,
    "successful": 1,
```

```
    "failed": 0
  },
  "created": true
}
```

Индекс создается, если он не существует, когда мы отправляем PUT-вызов.

Индексирование без идентификатора

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

Отклик:

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

Получение документов

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

Отклик:

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

Извлеките 10 документов из индекса `megacorp` с помощью `employee` типа:

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

Отклик:

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

```
  }
}
```

Простой поиск с помощью `match` запроса, который выглядит для точных соответствий в соответствующем поле:

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

Отклик:

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

Основные параметры поиска с примерами:

По умолчанию полный проиндексированный документ возвращается как часть всех запросов. Это называется (источник `_source` поле поиска хитов). Если мы не хотим, чтобы

весь исходный документ был возвращен, у нас есть возможность запросить только несколько полей из источника, которые будут возвращены, или мы можем установить `_source` на false, чтобы полностью опустить это поле.

В этом примере показано, как вернуть из поиска два поля: `account_number` и `balance` (внутри `_source`):

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

Обратите внимание, что приведенный выше пример просто уменьшает информацию, возвращаемую в поле `_source` . Он по-прежнему будет возвращать только одно поле с именем `_source` но будут включены только поля `account_number` и `balance` .

Если вы исходите из фона SQL, вышесказанное несколько похоже на концепцию SQL-запроса

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

Теперь перейдем к части запроса. Ранее мы видели, как запрос `match_all` используется для соответствия всем документам. Давайте теперь представим новый запрос, называемый запросом соответствия, который можно рассматривать как базовый полевой поисковый запрос (т. Е. Поиск, выполненный против определенного поля или набора полей).

В этом примере возвращается учетная запись с `account_number` установленным в `20` :

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

В этом примере возвращаются все учетные записи, содержащие термин «mill» в `address` :

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

В этом примере возвращаются все учетные записи, содержащие термин «mill» или «lane» в `address` :

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

Этот пример представляет собой вариант `match` (`match_phrase`), который разбивает запрос на термины и возвращает только документы, которые содержат все термины в `address` в тех же позициях относительно друг друга [1] .

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

Давайте теперь представим запрос bool (ean). Запрос bool позволяет нам составлять более мелкие запросы в более крупные запросы с использованием логической логики.

Этот пример содержит два совпадающих запроса и возвращает все учетные записи, содержащие «mill» и «lane» в адресе:

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

В приведенном выше примере условие bool `must` указывать все запросы, которые должны быть истинными для документа, считающегося совпадением.

Напротив, этот пример составляет два совпадающих запроса и возвращает все учетные записи, содержащие «mill» или «lane» в `address` :

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

В приведенном выше примере предложение bool `should` указывать список запросов, значение которых должно быть истинным для того, чтобы документ считался совпадением.

Этот пример составляет два совпадающих запроса и возвращает все учетные записи, которые не содержат ни «mill», ни «lane» в `address` :

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

В приведенном выше примере предложение bool must_not указывает список запросов, ни один из которых не должен быть истинным для того, чтобы документ считался совпадением.

Мы можем объединить пункты must, should и must_not одновременно внутри запроса bool. Кроме того, мы можем составлять запросы bool в любом из этих предложений bool, чтобы имитировать любую сложную многоуровневую логическую логику.

В этом примере возвращаются все учетные записи, принадлежащие тем, кому ровно 40 лет, и не живут в Вашингтоне (`WA` для краткости):

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

Установка Elasticsearch и Kibana на CentOS 7

Для запуска Elasticsearch на компьютере требуется Java Runtime Environment (JRE). Elasticsearch требует Java 7 или выше и рекомендует `Oracle JDK version 1.8.0_73` .

Поэтому убедитесь, что в вашей системе есть Java. Если нет, выполните следующие действия:

```
# 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 теперь должен быть установлен в вашей системе centOS. Вы можете проверить это с помощью:

```
java -version
```

Загрузить и установить 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
```

Запуск elasticsearch в качестве службы systemd при запуске

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

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

Установка Kibana

Первый импорт GPG-ключа на об / мин

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

Затем создайте локальный репозиторий `kibana.repo`

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

Добавьте следующий контент:

```
[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
```

Теперь установите кибану, выполнив команду:

```
yum -y install kibana
```

Начните с:

```
systemctl start kibana
```

Проверить статус с помощью:

```
systemctl status kibana
```

Вы можете запустить его в качестве службы запуска.

```
systemctl enable kibana
```

Прочитайте Начало работы с Elasticsearch онлайн:
https://riptutorial.com/ru/elasticsearch/topic/941/начало-работы-с-elasticsearch

глава 2: Анализаторы

замечания

Анализаторы берут текст из строкового поля и генерируют токены, которые будут использоваться при запросе.

Анализатор работает в последовательности:

- `CharFilters` (ноль или больше)
- `Tokenizer` (One)
- `TokenFilters` (ноль или больше)

Анализатор может быть применен к сопоставлениям, так что когда поля индексируются, это делается на основе каждого токена, а не на строке в целом. При запросе входная строка также будет запускаться через анализатор. Поэтому, если вы нормализуете текст в Анализаторе, он всегда будет соответствовать, даже если запрос содержит ненормированную строку.

Examples

картографирование

Анализатор может применяться к сопоставлению с помощью «анализатора», по умолчанию используется «стандартный» анализатор. В качестве альтернативы, если вы не хотите использовать какой-либо анализатор (поскольку токенизация или нормализация не были бы полезны), вы можете указать «index»: «not_analyzed»

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

Multi-поля

Иногда бывает полезно иметь несколько разных индексов поля с различными анализаторами. Вы можете использовать возможности нескольких полей для этого.

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

При запросе вместо простого использования «user.name» (который в этом случае все равно будет использовать анализатор stanard), вы можете использовать «user.name.special» или «user.name.unanalyzed». Обратите внимание, что документ останется без изменений, это влияет только на индексирование.

Анализаторы

Анализ в elasticsearch входит в контекст, когда вы готовы анализировать данные в своем индексе.

Анализаторы позволяют нам выполнять следующие действия:

- Сокращения
- Морфологический
- Типовая обработка

Теперь мы посмотрим на каждого из них.

1. **Сокращения** :

   Используя анализаторы, мы можем сказать elasticsearch, как обрабатывать аббревиатуры в наших данных, т. Е. Dr => Doctor, поэтому всякий раз, когда мы ищем ключевое слово врача в нашем индексе, elasticsearch также возвращает результаты, которые указаны в них.

2. **Стеблирование** :

   Использование стерилизации в анализаторах позволяет нам использовать базовые слова для модифицированных глаголов типа

слово	изменения
требовать	Требование, требуется

3. **Типовая обработка** :

   Анализаторы также обеспечивают обработку дескрипторов как при запросе, если мы ищем конкретное слово say 'resurrection', тогда elasticsearch вернет результаты, в которых присутствуют опечатки. Он будет обрабатывать опечатки, такие как resurection, reusurection, как таковые и будет перенастроить результат.

слово	изменения
воскрешение	Resurection, воскрешение

Анализаторы в Elasticsearch

1. стандарт
2. просто
3. Пробелы
4. Стоп
5. Ключевое слово
6. Шаблон
7. язык
8. Снежный шар

Игнорировать анализатор случаев

Иногда нам может потребоваться игнорировать случай нашего запроса относительно соответствия в документе. Анализатор можно использовать в этом случае, чтобы игнорировать случай во время поиска. Каждое поле должно содержать этот анализатор в свойстве, чтобы работать:

```
"settings": {
        "analysis": {
```

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

Прочитайте Анализаторы онлайн: https://riptutorial.com/ru/elasticsearch/topic/6232/анализаторы

глава 3: Изучение Elasticsearch с кибаной

Вступление

Kibana - это инструмент визуализации данных переднего конца для поиска elastics. для установки кибаны относятся к документации по кибане. Для запуска kibana на localhost перейдите в https: // localhost: 5601 и перейдите на консоль kibana.

Examples

Исследуйте свой кластер, используя Kibana

Синтаксис команды будет иметь следующий тип:

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

Выполните следующую команду для изучения кластера elasticsearch через Kibana Console.

- Для проверки работоспособности кластера

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

- Для перечисления всех индексов

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

- Для создания индекса с именем автомобиля

```
 PUT /car?pretty
```

- Для индексирования документа с именем автомобиля внешнего типа с использованием идентификатора 1

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

ответ вышеуказанного запроса будет:

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

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

- получение вышеуказанного документа может быть выполнено с использованием:

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

- Для удаления индекса

```
DELETE /car?pretty
```

Измените данные поиска elasticsearch

Elasticsearch обеспечивает возможности манипулирования данными и поиска данных практически в реальном времени. в этом примере у нас есть операции обновления, удаления и пакетной обработки.

- Обновление одного и того же документа. Предположим, мы уже проиндексировали документ на / car / external / 1. Затем запуск команды для индексирования данных заменяет предыдущий документ.

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

предыдущий документ автомобиля на id 1 с именем «Tata Nexon» будет обновлен с новым названием «Tata Nexa»,

- индексирование данных с явным идентификатором

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

  для индексации документа без Id мы используем глагол **POST** вместо глагола **PUT** . если мы не предоставим идентификатор, elasticsearch будет генерировать случайный идентификатор, а затем использовать его для индексации документа.

- Некорректное обновление предыдущего документа на Id.

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

- обновление документа с дополнительной информацией

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

- обновляя документ с помощью простых скриптов.

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

ctx._source ссылается на текущий исходный документ, который должен быть обновлен. Выше сценарий позволяет одновременно обновлять только один скрипт.

- Удаление документа

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

Примечание. Удаление всего индекса более эффективно, чем удаление всех документов с помощью команды «Удалить по запросу»

Пакетная обработка

Помимо индексации обновления и удаления документа, elasticsearch также предоставляет возможность выполнять любую из вышеперечисленных операций партиями с использованием API **_bulk** .

- для обновления нескольких документов с использованием API **_bulk**

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

- для обновления и удаления документов с использованием API **_bulk**

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

Если операция завершается неудачно, объемный API не прекращается. Он выполняет все операции и, наконец, возвращает отчет для всех операций.

Прочитайте Изучение Elasticsearch с кибаной онлайн:
https://riptutorial.com/ru/elasticsearch/topic/10058/изучение-elasticsearch-с-кибаной

глава 4: Интерфейс Python

параметры

параметр	подробности
хостов	Массив хостов в виде объектов , содержащих ключи `host` и `port` . По умолчанию `host` является «локальным» и `port` является 9200. Запись образца выглядит как `[{"host": "ip of es server", "port": 9200}]` - `[{"host": "ip of es server", "port": 9200}]` - `[{"host": "ip of es server", "port": 9200}]`
sniff_on_start	Boolean, если вы хотите, чтобы клиент обнюхивал узлы при запуске, sniffing означает получение списка узлов в кластере elasticsearch
sniff_on_connection_fail	Boolean для запуска sniffing, если соединение терпит неудачу, когда клиент активен
sniffer_timeout	разница во времени между секундами между каждым нюхом
sniff_timeout	время для одного запроса обнюхивания в секундах
retry_on_timeout	Booelan, если клиент должен тайм-аут, связавшись с другим узлом elasticsearch или просто выбросить ошибку
http_auth	Основная HTTP-аутентификация может быть представлена здесь в форме имени `username:password`

Examples

Индексирование документа (например, добавление образца)

Установите необходимую библиотеку Python через:

```
$ pip install elasticsearch
```

Подключитесь к Elasticsearch, создайте документ (например, ввод данных) и «Индекс» документа, используя 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"])
```

Подключение к кластеру

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

Создание пустого индекса и настройка отображения

В этом примере мы создаем пустой индекс (мы индексируем в нем никакие документы), определяя его отображение.

Сначала мы создаем экземпляр `ElasticSearch` а затем определяем отображение нашего выбора. Затем мы проверяем, существует ли индекс, а если нет, мы создаем его, указав параметры `index` и `body` которые содержат имя индекса и тело отображения, соответственно.

```
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))
```

Частичное обновление и обновление по запросу

Частичное обновление: используется, когда необходимо выполнить частичное обновление документа, то есть в следующем примере `name` поля документа с id `doc_id` будет обновляться до «John». Обратите внимание: если поле отсутствует, оно будет добавлено в документ.

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

Обновление по запросу: используется, когда необходимо обновить документы, удовлетворяющие условию, то есть в следующем примере мы обновляем возраст документов, чье поле `name` совпадает с «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')
```

Прочитайте Интерфейс Python онлайн: https://riptutorial.com/ru/elasticsearch/topic/2068/интерфейс-python

глава 5: кластер

замечания

Cluster Health предоставляет много информации о кластере, например количество выделенных («активных»), а также количество не назначенных и перемещаемых. Кроме того, он предоставляет текущее количество узлов и узлов данных в кластере, что позволяет вам опросить отсутствующие узлы (например, если вы ожидаете, что оно будет `15` , но оно отображает только `14` , то вам не хватает узла) ,

Для тех, кто знает об Elasticsearch, «назначенные» и «неназначенные» осколки могут помочь им выявить проблемы.

Наиболее распространенным полем, отмеченным в разделе « `status` кластера», является `status` , который может находиться в одном из трех состояний:

- красный
- желтый
- зеленый

Цвета каждого означают один - и только один - очень простая вещь:

1. Красный означает, что у вас отсутствует *хотя бы* один первичный осколок.
    - Отсутствующий первичный осколок означает, что индекс не может использоваться для записи (индексации) новых данных в большинстве случаев.
        - Технически вы можете индексировать все первичные осколки, доступные в этом индексе, но практически это означает, что вы не можете, потому что обычно не контролируете, какой осколок получает какой-либо документ.
        - Поиск по-прежнему возможен против красного кластера, но это означает, что вы получите частичные результаты, если какой-либо индекс, который вы ищете, не содержит осколков.
    - В обычных условиях это просто означает, что первичный осколок выделяется (`initializing_shards`).
    - Если узел просто покинул кластер (например, из-за того, что работающий на нем компьютер потерял электроэнергию), тогда имеет смысл, что вы *временно* потеряете некоторые первичные осколки.
        - Любой реплика-осколок для этого первичного осколка будет продвигаться как основной осколок в этом сценарии.
2. Желтый означает, что все первичные осколки активны, но *по крайней мере* один осколок реплики отсутствует.
    - Отсутствующая реплика влияет только на индексирование, если для параметров согласованности требуется, чтобы это повлияло на индексацию.
        - По умолчанию для любой первичной версии существует только одна

реплика, и индексирование может происходить с одной отсутствующей репликой.
    - В обычных обстоятельствах это просто означает, что выделяется осколок реплики (`initializing_shards`).
    - Один кластер узлов с включенными репликами *всегда* будет *в лучшем* случае желтым. Он может быть красным, если первичный осколок еще не назначен.
        - Если у вас только один узел, тогда имеет смысл отключить реплики, потому что вы не ожидаете. Тогда он может быть зеленым.
3. Зеленый означает, что все осколки активны.
    - Единственное действие наложения, которое разрешено для зеленого кластера, - это `relocating_shards` .
    - Новые индексы и, следовательно, новые осколки приведут к тому, что кластер переходит от красного к желтому до зеленого, так как каждый осколок выделяется (сначала первичный, делая его желтым, а затем реплики, если это возможно, делая его зеленым).
        - В Elasticsearch 5.x и более поздних версиях новые индексы **не** будут красить ваш кластер, если не потребуется слишком много времени для их выделения.

Examples

Человекочитаемое табличное кластерное здоровье с заголовками

В примере используется базовый синтаксис HTTP. Любые `<#>` в этом примере должны быть удалены при копировании.

Вы можете использовать API-интерфейсы `_cat` чтобы получить доступный для чтения, табличный вывод по различным причинам.

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

1. « `?v` является необязательным, но это означает, что вы хотите «подробный» вывод.

`_cat/health` существует с Elasticsearch 1.x, но вот пример его выхода из Elasticsearch 5.x:

С подробным выходом:

```
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%
```

Человеческое чтение, табличное здоровье кластера без заголовков

В примере используется базовый синтаксис HTTP. Любые `<#>` в этом примере должны быть

удалены при копировании.

Вы можете использовать API-интерфейсы `_cat` чтобы получить доступный для чтения, табличный вывод по различным причинам.

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

`_cat/health` существует с Elasticsearch 1.x, но вот пример его выхода из Elasticsearch 5.x:

Без подробного вывода:

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

Человеко-читаемое табличное кластерное здоровье с выбранными заголовками

В примере используется базовый синтаксис HTTP. Любые `<#>` в этом примере должны быть удалены при копировании.

Как и большинство API-интерфейсов `_cat` в Elasticsearch, API выборочно отвечает набором полей по умолчанию. Однако другие поля существуют из API, если вы хотите:

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

1. `?help` заставляет API возвращать поля (и короткие имена), а также краткое описание.

`_cat/health` существует с Elasticsearch 1.x, но вот пример его выхода из Elasticsearch 5.x:

Поля, доступные на момент создания этого примера:

```
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
```

Затем вы можете использовать это для печати только тех полей:

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

1. `h=...` определяет список полей, которые вы хотите вернуть.
2. `v` (verbose) определяет, что вы хотите, чтобы он печатал заголовки.

Результат из экземпляра Elasticsearch 5.x:

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

JSON-кластерное здоровье

В примере используется базовый синтаксис HTTP. Любые `<#>` в этом примере должны быть удалены при копировании.

API-интерфейсы `_cat` часто удобны для людей, чтобы получить подробные сведения о кластере. Но вы часто хотите, чтобы последовательно анализируемый вывод использовался с программным обеспечением. В общем, API JSON предназначены для этой цели.

```
GET /_cluster/health
```

`_cluster/health` существует с Elasticsearch 1.x, но вот пример его вывода из 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
}
```

Прочитайте кластер онлайн: https://riptutorial.com/ru/elasticsearch/topic/2069/кластер

глава 6: Команды завивки

Синтаксис

- curl -X <VERB> '<PROTOCOL>: // <HOST>: <PORT> / <PATH>? <QUERY_STRING>' -d '<BODY>'
- Куда:
- VERB: соответствующий HTTP-метод или глагол: GET, POST, PUT, HEAD или DELETE
- ПРОТОКОЛ: либо http, либо https (если у вас есть прокси-сервер https перед Elasticsearch).
- HOST: имя хоста любого узла в вашем кластере Elasticsearch или localhost для узла на вашем локальном компьютере.
- PORT: порт, на котором запущен HTTP-сервис Elasticsearch, который по умолчанию равен 9200.
- PATH: конечная точка API (например, _count вернет количество документов в кластере). Путь может содержать несколько компонентов, таких как _cluster / stats или _nodes / stats / jvm
- QUERY_STRING: любые необязательные параметры строки запроса (например, довольно красиво печатает ответ JSON, чтобы упростить его чтение).
- BODY: Тело запроса, закодированное JSON (если требуется запрос).
- Ссылка: Обсуждение с Elasticsearch: Elasticsearch Docs

Examples

Команда Curl для подсчета количества документов в кластере

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

Выход:

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

```
}
```

Индекс содержит 90 документов.

Ссылка: здесь

Получить документ по идентификатору

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

Выход:

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

Ссылка: здесь

Создать индекс

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

Выход:

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

Ссылка: здесь

Список всех индексов

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

выход:

```
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
```

Ссылка: здесь

Удалить индекс

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

выход:

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

Ссылка: здесь

Перечислить все документы в индексе

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

Это использует API `Search` и возвращает все записи под индексом `myIndexName` .

Ссылка: здесь

Прочитайте Команды завивки онлайн: https://riptutorial.com/ru/elasticsearch/topic/3703/команды-завивки

глава 7: Конфигурация Elasticsearch

замечания

Elasticsearch поставляется с набором параметров по умолчанию, которые обеспечивают хороший результат для разработки. Неявное утверждение заключается в том, что оно не обязательно отлично подходит для производства, которое должно быть адаптировано для ваших собственных нужд и поэтому не может быть предсказано.

Настройки по умолчанию позволяют легко загружать и запускать несколько узлов *на одном компьютере* без каких-либо изменений конфигурации.

Где настройки?

Внутри каждой установки Elasticsearch есть `config/elasticsearch.yml` . Вот где живут следующие настройки :

- `cluster.name`
  - Имя кластера, с которым соединяется узел. Все узлы в одном кластере **должны** иметь одинаковое имя.
  - В настоящее время по умолчанию используется `elasticsearch` .
- `node.*`
  - `node.name`
    - Если не указано, случайное имя будет генерироваться *каждый раз при запуске узла* . Это может быть весело, но это не хорошо для производственных сред.
    - Имена *не* обязательно должны быть уникальными, но они **должны** быть уникальными.
  - `node.master`
    - Логическая настройка. Когда `true` , то это означает , что узел имеет право на получение главного узла , и он может быть избран главный узел.
    - Значение по умолчанию равно `true` , что означает, что каждый узел является подходящим основным узлом.
  - `node.data`
    - Логическая настройка. Когда `true` , это означает, что узел хранит данные и обрабатывает активность поиска.
    - Значение по умолчанию равно `true` .
- `path.*`
  - `path.data`
    - Место, где файлы записываются для узла. *Все узлы используют этот каталог* для хранения метаданных, но узлы данных также используют его для хранения / индексирования документов.
    - По умолчанию `./data` .
      - Это означает, что `data` будут созданы для вас в качестве

одноранговой директории для `config` *внутри* каталога Elasticsearch.
    - `path.logs`
        - Место, где записываются файлы журнала.
        - По умолчанию. `./logs` .
- `network.*`
    - `network.host`
        - По умолчанию `_local_` , который является фактически `localhost` .
            - Это означает, что по умолчанию узлы не могут быть переданы извне текущего компьютера!
    - `network.bind_host`
        - Потенциально массив, это говорит Elasticsearch, какие адреса текущей машины также связывают сокеты.
            - Именно этот список позволяет машинам за пределами машины (например, другим узлам кластера) разговаривать с этим узлом.
        - По умолчанию используется `network.host` .
    - `network.publish_host`
        - Необычный хост, который используется для рекламы другим узлам, как наилучшим образом общаться с этим узлом.
            - При поставке массива в `network.bind_host` это должен быть *один* хост, предназначенный для межузловой связи.
        - По умолчанию используется network.host`.
- `discovery.zen.*`
    - `discovery.zen.minimum_master_nodes`
        - Определяет кворум для всеобщих выборов. Это **должно** быть установлено с использованием этого уравнения: `(M / 2) + 1` где `M` - количество *подходящих* основных узлов (узлы с использованием `node.master: true` неявно или явно).
        - По умолчанию `1` , что справедливо только для кластера с одним узлом!
    - `discovery.zen.ping.unicast.hosts`
        - Механизм присоединения этого узла к остальной части кластера.
        - Это *должно* отображать подходящие основные узлы, чтобы узел мог найти остальную часть кластера.
        - Значение, которое следует использовать здесь, это `network.publish_host` из этих других узлов.
        - По умолчанию используется `localhost` , что означает, что он смотрит только на локальный компьютер для объединения кластера.

Какие существуют настройки?

Elasticsearch предоставляет три различных типа настроек:

- Настройки кластера
    - Это настройки, которые применяются ко всему в кластере, такие как все узлы или все индексы.
- Настройки узла
    - Это настройки, которые применяются только к текущему узлу.
- Настройки индекса
    - Это настройки, которые относятся только к индексу.

В зависимости от настройки, это может быть:

- Изменено динамически во время выполнения
- Изменен после перезапуска (закрытие / открытие) индекса
    - Некоторые настройки уровня индекса не требуют, чтобы индекс был закрыт и снова открыт, но может потребовать, чтобы индекс был принудительно повторно объединен для применяемого параметра.
        - Примером такого типа настроек является уровень сжатия индекса. Его можно изменять динамически, но только новые *сегменты* используют это изменение. Поэтому, если индекс не изменится, он никогда не воспользуется этим изменением, если вы не заставите индекс воссоздать свои сегменты.
- Изменено после перезапуска узла
- Изменен после перезапуска кластера
- Никогда не менялся

Всегда проверяйте документацию для своей версии Elasticsearch на то, что вы можете или не можете сделать с настройкой.

Как я могу применить настройки?

Вы можете установить настройки несколькими способами, некоторые из которых не предлагаются:

- Аргументы командной строки

В Elasticsearch 1.x и 2.x вы можете отправить большинство настроек в качестве свойств Java System с префиксом `es.` :

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

В Elasticsearch 5.x это изменяется, чтобы избежать использования свойств Java-системы, вместо этого используя собственный тип аргумента с `-E` , `-Des.` :

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

Такой подход к применению настроек отлично работает при использовании таких инструментов, как Puppet, Chef или Ansible для запуска и остановки кластера. Однако при работе он работает очень плохо.

- Настройки YAML
  - Показаны в примерах
- Динамические настройки
  - Показаны в примерах

Порядок, в котором применяются настройки, находится в порядке наибольшей динамичности:

1. Временные настройки
2. Постоянная настройка
3. Настройки командной строки
4. Настройки YAML (статические)

Если настройка установлена дважды, один раз на любом из этих уровней, то самый высокий уровень вступает в силу.

Examples

Настройки статического эластинга

Elasticsearch использует файл конфигурации YAML (еще один язык разметки), который можно найти внутри каталога Elasticsearch по умолчанию (установки RPM и DEB меняют это местоположение между прочим).

Вы можете установить основные настройки в `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_
```

Постоянные настройки динамического кластера

Если вам нужно применить настройку динамически после того, как кластер уже запущен, и его можно установить динамически, вы можете установить его с `_cluster/settings` API-интерфейса `_cluster/settings` .

Постоянными настройками являются один из двух типов кластерного уровня, которые могут применяться. Устойчивая установка **выживет** полный перезапуск кластера.

Примечание. Не все настройки могут применяться динамически. Например, имя кластера не может быть переименовано динамически. Большинство настроек уровня узла нельзя установить динамически (поскольку они не могут быть нацелены индивидуально).

Это **не** API для установки параметров уровня индекса. Вы можете сказать, что это параметр уровня индекса, потому что он должен начинаться с `index.` , Настройки, имена которых указаны в виде `indices.` *являются параметрами* кластера, поскольку они применяются ко всем индексам.

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

Предупреждение . В Elasticsearch 1.x и 2.x вы не можете *отключить* постоянную настройку.

К счастью, это улучшилось в Elasticsearch 5.x, и теперь вы можете удалить параметр, установив его в `null` :

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

Неустановленная настройка вернется к своему значению по умолчанию или к любому значению, заданному на более низком уровне приоритета (например, настройке командной строки).

Переходные динамические настройки кластера

Если вам нужно применить настройку динамически после того, как кластер уже запущен, и его можно установить динамически, вы можете установить его с `_cluster/settings` API-интерфейса `_cluster/settings` .

Параметры переходного процесса являются одним из двух типов кластерного уровня, которые могут применяться. Параметр переходного процесса **не** сможет полностью перезагрузить кластер.

Примечание. Не все настройки могут применяться динамически. Например, имя кластера не может быть переименовано динамически. Большинство настроек уровня узла нельзя установить динамически (поскольку они не могут быть нацелены индивидуально).

Это **не** API для установки параметров уровня индекса. Вы можете сказать, что это параметр уровня индекса, потому что он должен начинаться с `index.` , Настройки, имена которых указаны в виде `indices.` *являются параметрами* кластера, поскольку они применяются ко всем индексам.

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

Предупреждение . В Elasticsearch 1.x и 2.x вы не можете отключить настройки переходного процесса без полного перезапуска кластера.

К счастью, это улучшилось в Elasticsearch 5.x, и теперь вы можете удалить параметр, установив его в null:

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

Неустановленная настройка вернется к своему значению по умолчанию или любому значению, заданному на более низком уровне приоритета (например, `persistent` настройкам).

Настройки индекса

Параметры индекса - это те настройки, которые относятся к одному индексу. Такие настройки начинаются с `index.` , Исключением из этого правила являются `number_of_shards` и `number_of_replicas` , которые также существуют в виде `index.number_of_shards` и `index.number_of_replicas` .

Как следует из названия, настройки уровня индекса применяются к одному индексу. Некоторые настройки должны применяться во время создания, потому что они не могут быть изменены динамически, например параметр `index.number_of_shards` , который контролирует количество первичных обрывов для индекса.

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

или, в более сжатом формате, вы можете комбинировать ключевые префиксы в каждом . :

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

В приведенных выше примерах будет создан индекс с предоставленными настройками. Вы можете динамически изменять настройки для каждого индекса, используя `_settings` точку `_settings` . Например, здесь мы динамически изменяем настройки медленного режима *только* для уровня предупреждения:

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

Предупреждение : Elasticsearch 1.x и 2.x не очень строго проверяли имена установок уровня индекса. Если у вас была опечатка или просто составлена настройка, тогда она слепо приняла бы ее, но в противном случае игнорировала бы ее. Elasticsearch 5.x строго проверяет имена параметров и отклоняет любую попытку применить параметры индекса с неизвестными настройками (из-за опечатки или отсутствующего плагина). Оба утверждения относятся к динамически изменяющимся настройкам индекса и во время создания.

Динамические параметры индекса для нескольких индексов одновременно

Вы можете применить те же изменения, что указаны в примере « `Index Settings` ко *всем* существующим индексам с одним запросом или даже подмножеством из них:

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

или же

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

или же

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

Если вы предпочитаете более выборочно делать это, то вы можете выбрать несколько без подачи всех:

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

Прочитайте Конфигурация Elasticsearch онлайн:
https://riptutorial.com/ru/elasticsearch/topic/3411/конфигурация-elasticsearch

глава 8: Поиск API

Вступление

API поиска позволяет выполнять поисковый запрос и возвращать поисковые запросы, соответствующие запросу. Запрос может быть предоставлен с использованием простой строки запроса в качестве параметра или с использованием тела запроса.

Examples

маршрутизация

При выполнении поиска он будет транслироваться на все индексы / индексы (круговое разворот между репликами). Какими осколками будет выполняться поиск, можно управлять с помощью параметра маршрутизации. Например, при индексировании твитов значением маршрутизации может быть имя пользователя:

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

Поиск с использованием тела запроса

Поиски также могут выполняться на elasticsearch с использованием поискового DSL. Элемент запроса в теле запроса поиска позволяет определить запрос с использованием DSL запроса.

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

Мульти-поиск

Параметр multi_search позволяет нам искать запрос в нескольких полях одновременно.

```
GET /_search
{
  "query": {
    "multi_match" : {
      "query":    "text to search",
```

```
      "fields": [ "field_1", "field_2" ]
    }
  }
}
```

Мы также можем увеличить баллы определенных полей с помощью оператора boost (^) и использовать в поле имени (*) дикие карты,

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

Поиск в URI и выделение

Запрос поиска может быть выполнен с использованием URI, предоставляя параметры запроса. Не все параметры поиска отображаются при выполнении поиска в этом режиме, но это может быть удобно для быстрого «скручивания».

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

Еще одна полезная функция - выделение совпадений в документах.

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

В приведенном выше случае конкретное поле будет выделено для каждого поиска

Прочитайте Поиск API онлайн: https://riptutorial.com/ru/elasticsearch/topic/8625/поиск-api

глава 9: Разница между индексами и типами

замечания

Легко видеть `type` s как таблицу в базе данных SQL, где `index` - это база данных SQL. Однако это не очень хороший подход к `type` s.

Все о типах

Фактически, типы - это *буквально* только поле метаданных, добавленное в каждый документ `_type` : `_type` . В приведенных выше примерах создаются два типа: `my_type` и `my_other_type` . Это означает, что каждый документ, связанный с типами, имеет дополнительное поле, автоматически определяемое как `"_type": "my_type"` ; это индексируется с документом, что делает его *полем поиска или фильтрации* , но оно не влияет на сам исходный документ, поэтому вашему приложению не нужно беспокоиться об этом.

Все типы живут в одном и том же индексе и, следовательно, в тех же коллективных осколках индекса. Даже на уровне диска они живут в одних и тех же файлах. Единственное разделение, создающее второй тип, является логичным. Каждый тип, независимо от того, является он уникальным или нет, должен существовать в сопоставлениях, и все эти сопоставления должны существовать в вашем состоянии кластера. Это поглощает память, и, если каждый тип обновляется динамически, он увеличивает производительность при изменении сопоставлений.

Таким образом, лучше всего определить только один тип, если вам действительно не нужны другие типы. Обычно встречаются сценарии, в которых желательны несколько типов. Например, представьте, что у вас был автомобильный указатель. Возможно, вам будет полезно разбить его несколькими типами:

- БМВ
- гнаться
- Хонда
- мазда
- мерседес
- ниссан
- Range Rover
- Тойота
- ...

Таким образом, вы можете искать все автомобили или ограничивать их производителем по требованию. Разница между этими двумя поисками также проста:

```
GET /cars/_search
```

а также

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

Что не очевидно для новых пользователей Elasticsearch, так это то, что вторая форма является специализацией первой формы. Он буквально переписывается:

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

Он просто отфильтровывает любой документ, который не был проиндексирован в поле `_type` , значение которого было `bmw` . Поскольку каждый документ индексируется с его типом в качестве поля `_type` , это служит довольно простым фильтром. Если фактический поиск был предоставлен в любом примере, тогда фильтр будет добавлен к полному поиску, если это необходимо.

Таким образом, если типы идентичны, гораздо лучше поставлять один тип (например, `manufacturer` в этом примере) и эффективно игнорировать его. Затем в каждом документе явно укажите поле под названием `make` *или другое имя, которое вы предпочитаете,* и вручную фильтруйте его, когда вы хотите его ограничить. Это уменьшит размер ваших сопоставлений до `1/n` где `n` - количество отдельных типов. Он добавляет другое поле к каждому документу в пользу упрощенного сопоставления.

В Elasticsearch 1.x и 2.x такое поле должно быть определено как

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

```
    }
  }
}
```

1. Имя произвольное.
2. Имя произвольное, *и* оно может совпадать с именем типа, если вы тоже этого хотели.

В Elasticsearch 5.x вышеупомянутое все равно будет работать (оно устарело), но лучший способ - использовать:

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

1. Имя произвольное.
2. Имя произвольное, *и* оно может совпадать с именем типа, если вы тоже этого хотели.

Типы должны использоваться экономно в пределах ваших индексов, потому что они раздувают сопоставления индексов, как правило, без особых преимуществ. У вас должен быть хотя бы один, но нет ничего, что говорит, что у вас должно быть больше одного.

Общие вопросы

- Что делать, если у меня есть два (или более) типа, которые в основном идентичны, но которые имеют несколько уникальных полей для каждого типа?

На уровне индекса нет разницы между одним типом, который используется с несколькими полями, которые редко используются, *и* между несколькими типами, которые разделяют кучу не разреженных полей с несколькими не разделяемыми (что означает, что другой тип никогда не использует поле (ы)).

Сказано иначе: редко используемое поле разрежено по индексу *независимо от типов* . Разделение не приносит пользу - или действительно больно - индекс только потому, что он определен в отдельном типе.

Вы должны просто объединить эти типы и добавить отдельное поле типа.

- Почему отдельные типы должны точно определять поля точно так же?

Поскольку каждое поле действительно определено только один раз на уровне Lucene, независимо от количества типов. Тот факт, что типы существуют вообще, является

особенностью Elasticsearch, и это *только* логическое разделение.

- Могу ли я определить отдельные типы с одинаковым полем, определенным по-разному?

Нет. Если вам удастся найти способ сделать это в ES 2.x или новее, тогда вы должны открыть отчет об ошибке . Как отмечалось в предыдущем вопросе, Луцен видит их всех как одно поле, поэтому нет возможности сделать эту работу надлежащим образом.

ES 1.x оставил это как неявное требование, которое позволило пользователям создавать условия, при которых одно отображение осколков в индексе фактически отличалось от другого осколка в том же индексе. Это было фактически условием гонки, и это *могло* привести к неожиданным проблемам.

Исключения из правила

- Родительские / дочерние документы **требуют использования** отдельных типов в одном и том же индексе.
  - Родитель живет одним типом.
  - Ребенок живет в отдельном типе (но каждый ребенок живет в том же *осколке,* что и его родитель).
- Чрезвычайно нишевые случаи использования, когда создание тонны индексов нежелательно, а влияние разреженных полей предпочтительнее альтернативы.
  - Например, плагин мониторинга Elasticsearch, Marvel (1.x и 2.x) или X-Pack Monitoring (5.x +), контролирует сам Elasticsearch для изменений в кластере, узлах, индексах, конкретных индексах (уровне индекса), и даже осколки. Он может создавать 5 + индексов каждый день, чтобы изолировать те документы, которые имеют уникальные сопоставления, *или* может пойти против лучших практик, чтобы уменьшить нагрузку на кластер, разделив индекс (обратите внимание: количество определенных сопоставлений фактически одинаково, но количество созданных индексов уменьшается от `n` до 1).
  - Это расширенный сценарий, но вы должны учитывать общие определения полей для разных типов!

Examples

Явное создание индекса с типом

В примере используется базовый HTTP, который легко преобразуется в cURL и другие HTTP-приложения. Они также соответствуют синтаксису Sense , который будет переименован в Console в Kibana 5.0.

Примечание. В примере добавляется `<#>` чтобы привлечь внимание к частям. Их следует удалить, если вы его скопируете!

```
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. Это создает `index` используя конечную точку создания индекса.
2. Это создает `type` .
3. Общие поля `type` s внутри одного и того же `index` **должны** иметь одно и то же определение! ES 1.x не строго соблюдал это поведение, но это было неявное требование. ES 2.x и выше строго соблюдают это поведение.
4. Уникальные поля `type` s в порядке.

Индексы (или индексы) *содержат* типы. Типы - удобный механизм для разделения документов, но они требуют, чтобы вы определяли - либо динамически / автоматически, либо явно - сопоставление для каждого используемого вами типа. Если вы определяете 15 типов в индексе, то у вас есть 15 уникальных сопоставлений.

См. Замечания для получения более подробной информации об этой концепции и почему вы можете или не хотите использовать типы.

Динамическое создание индекса с типом

В примере используется базовый HTTP, который легко преобразуется в cURL и другие HTTP-приложения. Они также соответствуют синтаксису Sense , который будет

переименован в Console в Kibana 5.0.

Примечание. В примере добавляется `<#>` чтобы привлечь внимание к частям. Их следует удалить, если вы его скопируете!

```
DELETE /my_index <1>

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

1. Если он уже существует (из-за более раннего примера), удалите индекс.
2. Индексируйте документ в индекс `my_index` с типом, `my_type` и ID `abc123` (может быть числовым, но это всегда строка).
    - По умолчанию динамическое создание индекса включается простым индексированием документа. Это отлично подходит для сред разработки, но это не обязательно хорошо для производственных сред.
3. Это поле является целым числом, поэтому при первом просмотре его необходимо сопоставить. Elasticsearch всегда принимает самый *широкий* тип для любого входящего типа, поэтому он будет отображаться как `long` а не `integer` или `short` (оба из которых могут содержать `1234` и `456`).
4. То же самое верно и для этого поля. Он будет отображаться как `double` а не как `float` как вы, возможно, захотите.

Этот динамически созданный индекс и тип грубо соответствуют сопоставлению, определенному в первом примере. Однако очень важно понять, как `<3>` и `<4>` влияют на автоматически определенные сопоставления.

Вы могли бы следовать этому, добавив еще один тип динамически к одному и тому же индексу:

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

1. Тип является единственным отличием от вышеуказанного документа. Идентификатор тот же, и все в порядке! Это не имеет никакого отношения к другим `abc123` кроме того, что это *происходит* с тем же индексом.
2. `field1` уже существует в индексе, поэтому он *должен* быть тем же типом поля, что и в других типах. Отказ от значения, который был строкой или не целой, потерпел бы неудачу (например, `"field1": "this is some text"` или `"field1": 123.0`).

Это будет динамически создавать сопоставления для `my_other_type` в одном и том же индексе `my_index` .

Примечание. *Всегда* быстрее определять сопоставления, вместо того, чтобы Elasticsearch динамически выполнял его во время индекса.

Конечный результат индексации обоих документов будет аналогичен первому примеру, но типы полей будут разными и, следовательно, немного расточительными:

```
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. Это использует `_mappings` точку `_mappings` чтобы получить сопоставления из созданного нами индекса.
2. Мы динамически создали `my_type` на первом этапе этого примера.
3. `field2` теперь `long` а не `integer` потому что мы не определяли его заранее. Это *может* оказаться расточительным в дисковой памяти.
4. `object1.field1` теперь является `double` по той же причине, что и # 3 с теми же последствиями, что и # 3.
    - Технически, `long` может быть сжат во многих случаях. Однако `double` нельзя сжать из-за того, что он является числом с плавающей запятой.
5. Мы также динамически создали `my_other_type` на втором этапе этого примера. Его

сопоставление происходит одинаково, потому что мы уже использовали `long` и `double` .

- Помните, что `field1` *должно* соответствовать определению из `my_type` (и оно).
- `field3` уникально для этого типа, поэтому оно не имеет такого ограничения.

Прочитайте Разница между индексами и типами онлайн:
https://riptutorial.com/ru/elasticsearch/topic/3412/разница-между-индексами-и-типами

глава 10: Разница между реляционными базами данных и Elasticsearch

Вступление

Это для читателей, которые исходят из реляционного фона и хотят изучить elasticsearch. В этом разделе показаны примеры использования, для которых реляционные базы данных не являются подходящим вариантом.

Examples

Разница терминологии

Реляционная база данных	Elasticsearch
База данных	Индекс
Таблица	Тип
Строка / запись	Документ
Название столбца	поле

Над таблицей грубо рисуется аналогия между базовыми элементами реляционной базы данных и elasticsearch.

Настроить

Учитывая следующую структуру в реляционной базе данных:

```
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 Эквивалент:

```
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, где реляционные базы данных не подходят

- Суть поиска лежит в его порядке. Каждый хочет, чтобы результаты поиска отображались таким образом, чтобы наилучшие результаты были показаны сверху. Реляционная база данных не имеет такой возможности. С другой стороны, Elasticsearch показывает результаты на основе релевантности по умолчанию.

  Настроить

  То же, что используется в предыдущем примере.

  Постановка задачи

Предположим, что пользователь хочет найти `shirts` но он заинтересован в `red` рубашках. В этом случае результаты, содержащие ключевое слово `red` и `shirts` должны быть сверху. Затем после них следует показывать результаты для других рубашек.

Решение с использованием запроса реляционной базы данных

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

Выход

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

Решение Elasticsearch

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

Выход

```
"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"
     }
  }
 ]
```

Заключение

Как мы видим выше, реляционная база данных вернула результаты в некотором

случайном порядке, в то время как Elasticsearch возвращает результаты в порядке убывания `_score` который рассчитывается на основе релевантности.

- Мы склонны ошибаться при вводе строки поиска. Бывают случаи, когда пользователь вводит неверный параметр поиска. Реляционные базы данных не будут обрабатывать такие случаи. Эластичный поиск на помощь.

  Настроить

  То же, что используется в предыдущем примере.

  Постановка задачи

  Предположим , что *пользователь хочет найти* `shirts` , *но он входит в неправильное слово* `shrt` *по ошибке. Пользователь все еще ожидает увидеть результаты рубашки* .

  Решение с использованием запроса реляционной базы данных

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

  Выход

  ```
  No results found
  ```

  Решение Elasticsearch

  ```
  POST /test/product/_search

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

  Выход

  ```
  "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"
      }
   }
]
```

Заключение

Как мы видим выше, реляционная база данных не вернула результатов поиска неправильного слова, в то время как Elasticsearch с использованием своего специального `fuzzy` запроса возвращает результаты.

Прочитайте Разница между реляционными базами данных и Elasticsearch онлайн: https://riptutorial.com/ru/elasticsearch/topic/10632/разница-между-реляционными-базами-данных-и-elasticsearch

глава 11: Скопления

Синтаксис

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

Examples

Среднее агрегирование

Это агрегирование агрегатов с одним значением, которое вычисляет среднее число числовых значений, которые извлекаются из агрегированных документов.

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

Вышеуказанное агрегирование вычисляет средний класс по всем документам. Тип агрегации - avg, а параметр поля определяет числовое поле документов, на которое рассчитывается среднее значение. Вышеупомянутое вернет следующее:

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

Имя агрегации (avg_grade выше) также служит ключом, по которому результат агрегирования может быть получен из возвращаемого ответа.

Агрегация мощности

Единичная агрегация показателей, которая вычисляет приблизительное количество различных значений. Значения могут быть извлечены либо из определенных полей документа, либо сгенерированы скриптом.

```
POST /index/_search?size=0
{
```

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

Отклик:

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

Расширенная статистика агрегирования

Агрегирование многозначных показателей, которое вычисляет статистику по числовым значениям, извлеченным из агрегированных документов. Эти значения могут быть извлечены либо из определенных числовых полей в документах, либо сгенерированы предоставленным сценарием.

Агрегаты extended_stats - это расширенная версия агрегации статистики, в которую добавляются дополнительные метрики, такие как sum_of_squares, variance, std_deviation и std_deviation_bounds.

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

Пример вывода:

```
{
    ...

    "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
          }
        }
    }
}
```

Прочитайте Скопления онлайн: https://riptutorial.com/ru/elasticsearch/topic/10745/скопления

кредиты

S. No	Главы	Contributors
1	Начало работы с 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	Анализаторы	Bhushan Gadekar, Sid1199, Thomas
3	Изучение Elasticsearch с кибаной	sarvajeetsuman
4	Интерфейс Python	aidan.plenert.macdonald, christinabo, KartikKannapur, pickypg, Sumit Kumar
5	кластер	Gerardo Rochín, pickypg
6	Команды завивки	Fawix, Mrunal Pagnis, Mrunal Pagnis
7	Конфигурация Elasticsearch	pickypg
8	Поиск API	aerokite, Sid1199
9	Разница между индексами и типами	pickypg
10	Разница между реляционными базами данных и Elasticsearch	Richa
11	Скопления	Sid1199