ELASTICSEARCH   
The definite Guide

Chapter1

ElasticSearch is also written in Java and uses Lucene internaly.

ElassticSearch is:

* A distributed real-time document store where every field is indexed and seachable.
* A distributed search engine with real-time analytic.
* A capable of scalling to hundreds of servers and petabytes of structured and unstructured data.

Package all of this functionality into a standalone server that your application can talk to viaa simple RESTful API, using web client from your favourite programminf language or even from command line.

Clusters and nodes

Cluster - is a group of nodes with the same name(*cluster.name*), which are working togather.

Node - is a running instance of ElasticSearch.

*Change cluste.name -> elasticsearch.yml*

*Вимкнути кластер: POST \_shutdown АБО ctrl+C*

Java API

If you are using Java, then ElasticSearch comes with two built-in clients which you can use in your code.

*Node client*

Join a local cluster as a *none-data node.* In other worlds is doesnt hold any data itself, it knows where data lives on which node in the cluster and can forward request directly to the correct node.

*(Клієнт приэднуэться до кластера, як none-data node. Іншими словами, він не зберігає ніяких даних сам,але він знає в якій ноді на кластері знаходяться дані і може ПРИШВИДШУВАТИ ЗАПИТИ, безпосередньо на підходящий нод.)*

*Transport client*

Can be used to send requests to a remote cluster. It doesnt join the cluster itself, but simply forwards requests to a remote cluster. It doesnt join the cluster itself, but simply forwards requests to a node in a cluster.

! Both Java clients talk to the cluster over port 9300, using native ElasticSearch *transport protocol*.

The nodes in the cluster also communicate with each other over port 9300. If this port is not open, then youe nodes will not be able to form a cluster.

Java client must be from the same version of ElasticSearcr as the nodes, oterwise thay may not b able to understnd each other.

RESTful API with JSON over HTTP

All the other languages can communicate with Elastisearch over port 9200 using RESTful API.

Elastics provides official clients for several languages - Gorrvy, Javascript, .NET, PHP, Perl, Python, and Ruby.

To count number of document on a cluster:

*(Скільки всього документів в кластері?)*

GET /\_count

{

“query”:{

“match\_all”:{}

}

}

JSON

JSON supported by most programming languages and has become the standart format used by NoSql databases.

Converting your data to JSON

Almost all languages have modules to convert data structures or objects into JSON. The official ElasticSearch clients all handle conversion to and from JSON for you automatically.

Elasticsearch basic concepts:

-indexing

- search

- aggregations

Indexing tha data: Relational DB vs Elasticsearch

Relational DB: Databases -> Tables -> Rows -> Columns

Elasticsearch: Indices -> Types -> Documents -> Fields

Index document = to store document (in relational databases)

Relation database add in *index* to specific columns in order to improve the speed of data retrieval.

ElasticSearch use a structure called an *inrevted index* for exactly the same purpose.

By default every field in a document is indexed and thus are searchable. A field without inverted index is not searchable.

Work with document

To replace an existing document with update version, we just PUT it again.

Search

GET /index/type/\_search -------> By default search return top 10 results.

1. *Add some employee to megacorp company:*

FIRST DATA FOR EXAMPLES:

PUT /megacorp/employee/1

{

"first\_name": "John",

"last\_name" : "Smith",

"age" : 25,

"about" : "I love to go rock climbing",

"interests" : ["sports", "music"]

}

PUT /megacorp/employee/2

{

"first\_name": "Jane",

"last\_name" : "Smith",

"age" : 32,

"about" : "I like to correct rock albums",

"interests" : ["music"]

}

PUT /megacorp/employee/3

{

"first\_name": "Douglas",

"last\_name" : "Fir",

"age" : 35,

"about" : "I love to build cabinets",

"interests" : ["forestry"]

}

*! To replace exist document use you can just PUT it again.*

\_Search

GET /megacorp/employee/1 -get all information about employee1

First simple search

*Get all emloyees with this request:*

*GET /megacorp/employee/\_search*

*!By default, request return the top 10 results.*

“lightweight” search - easy to use in a command line:

*GET /megacorp/employee/\_search?q=last\_name:Smith*

*the full right request by Ksu:*

*curl -GET 'localhost:9200/megacorp/employee/\_search?q=last\_name:Smith'*

Search with query DSL

ElasticSearch provides a rich, flexible query language called Query DSL(Domain Specific Language).

*Example of match query:*

*GET /megacorp/employee/\_search*

*{*

*"query":{*

*"match": {*

*"last\_name": "Smith"*

*}*

*}*

*}*

More complicated search

Example:

GET /megacorp/employee/\_search

{

"query":{

"filtered": {

"query": {},

"filter": {}

}

}

}

*We still wat to find employee with a last name of “Smith”, but we only want employees who are older than 30:*

GET /megacorp/employee/\_search

{

"query":{

"filtered": {

"query": {

"match": {

"last\_name": "smith"

}

},

"filter": {

"range": {

"age": {

"gt": 30

}

}

}

}

}

}

Full-text search - *!Пошук по фразі, але при цьому аналізуються слова фрази і тоді шукається співпадання. Порядок слів у фразі не аналізується.*

The searches so far have been simple: single names, filtering by age. Let’s try a more advanced, full-text search - a task which traditional databases would really struggle with.

*Example2.*

*We are going to search employees who really enjoy “rock climbing”:*



GET /megacorp/employee/\_search

{

"query": {

"match": {

"about": "rock climbing"

}

}

}

*! By default Elastic sorts matching results by their relevance score, that is: by how well each document matched the query. The first and highest scorind result is obvious(очевидний).*

Phrase search - *!Пошук по словах, що ідуть підряд у реченні, тобто враховується порядок слів.*

*!Тобто коли важливо знайти точний пордок слів.*

*Example3.*

*We are going to search employees that contain both “rock” and ”climbing”and where the words are next to each other = “rock climbing”:*

*GET megacorp/employee/\_search*

*{*

*"query": {*

*"match\_phrase": {*

*"about": "rock climbing"*

*}*

*}*

*}*

Highlighting of searches - *підсвідка слів, які спідпадають з тими, які шукаємо.*

Many application like to highlight snippets of text from each search result so that the user can see why the documents matched their query. Retrieving highlighted fragments is very easy in ElasticSearch.

*Example4.*

*Lets rerun the previous query, but add a new highlight parametr:*

*(До попереднього запросу додаємо параметр "highlight")*

GET megacorp/employee/\_search

{ "query": {

"match\_phrase": {

"about": "rock climbing"

}

},

"highlight": {

"fields": {

"about": {}

}

}

}

! Query return the same results, but now we get a new section in the response called highlight. this contain a snippet of text from the about field with the matching words wrapped in [<em></em>](http://htmlbook.ru/html/em) HTML tags.

You can read more about the highlighting of search snippets in the

Analytics

ElasticSearch has functionality called aggregation, which allows you to generate sophisticated analytics over your data. It as similar to GROUP BY in SQL, but much more powerful.

*Example5.*

*Let’s find the most popular interests enjoyed by our employees:*

*general view:*

*GET /megacorp/employee/\_search*

*{*

*"aggs": {*

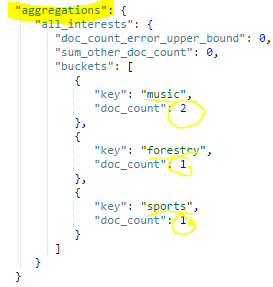
*"NAME": {*

*"AGG\_TYPE": {}*

*}*

*}*

*}*



GET /megacorp/employee/\_search

{

"aggs": {

"all\_interests": {

"terms": {

"field": "interests"

}

}

}

}

*Fields:*

*\*all\_interests - aggregation name, may be any you want*

*\* terms - kind of aggregation*

*Example6 (example5+ last\_name=smith)*

*Let’s find the most popular interests enjoyed by our employees, but with last\_name = “smith”:*

GET /megacorp/employee/\_search

{ "query": {

"match": {

"last\_name": "smith"

}

},

"aggs": {

"all\_interests": {

"terms": {

"field": "interests"

}

}

} }

The sky is the limit as to what kind of data you can extract!