

- [Agile](#)
- [Architecture](#)
- [ECM](#)
- [Java](#)
- [Performance](#)
- [Homepage](#)



Elasticsearch 101

February 7, 2014 by [dennis.probst](#)

Introduction

Elasticsearch is a highly scalable search engine that stores data in a structure optimized for language based searches and it is a whole lot of fun to work with. In this 101 I'll give you a hands-on introduction to Elasticsearch and give you a glimpse at some of the key concepts.

- open-source
- distributed: clustering, replication, fail over, and master election out of the box
- schema less: document based, automated type mapping, JSON
- RESTful API
- highly configurable
- sane defaults
- data exploration
- based on Lucene
- runs on the JVM
- fast

Running Elasticsearch

Elasticsearch is a standalone Java application, so getting up and running is a piece of cake. Make sure you have Java $\geq 1.6.0$ and that no one else is running Elasticsearch in your network. You can either download a packaged distribution from elasticsearch.org and unpack it or get the latest source from [github](https://github.com).

Start a node in foreground with

```
$ bin/elasticsearch -f
```

You should see something like this

```

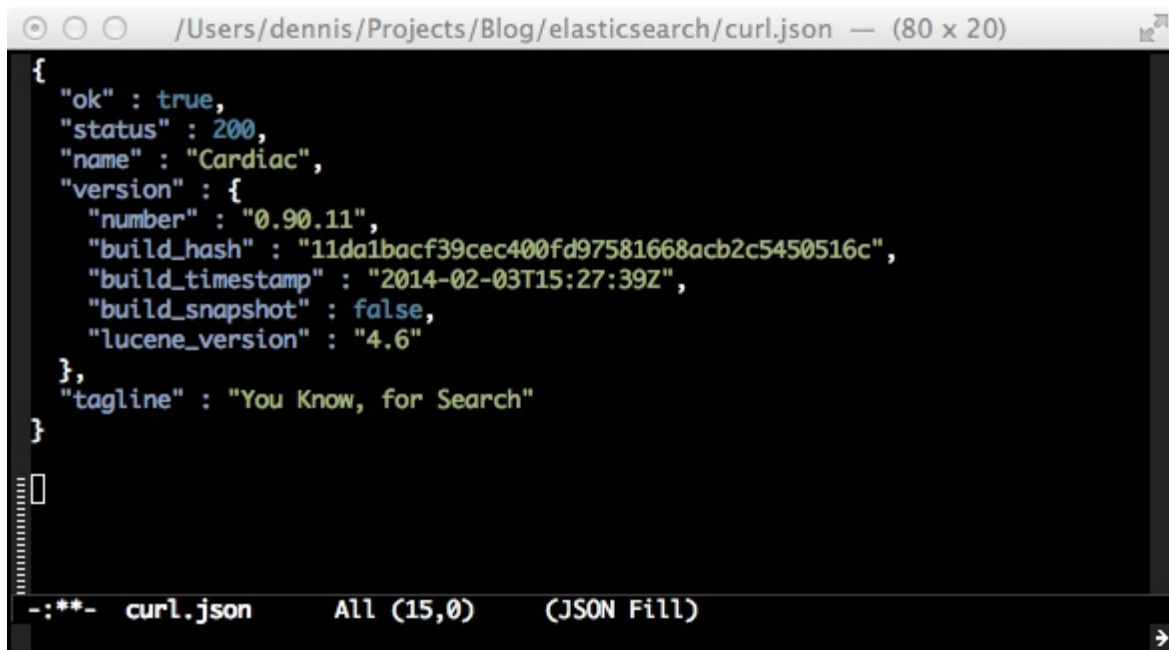
Ben:elasticsearch-0.90.11 dennis$ bin/elasticsearch -f
[2014-02-04 09:36:50,652][INFO ][node                ] [Lorna Dane] version[0.90.11], pid[7202]
, build[11da1ba/2014-02-03T15:27:39Z]
[2014-02-04 09:36:50,654][INFO ][node                ] [Lorna Dane] initializing ...
[2014-02-04 09:36:50,666][INFO ][plugins          ] [Lorna Dane] loaded [], sites []
[2014-02-04 09:36:54,516][INFO ][node                ] [Lorna Dane] initialized
[2014-02-04 09:36:54,517][INFO ][node                ] [Lorna Dane] starting ...
[2014-02-04 09:36:54,729][INFO ][transport        ] [Lorna Dane] bound_address {inet[/0:0:0:
0:0:0:0:9300]}, publish_address {inet[/192.168.101.19:9300]}
[2014-02-04 09:36:57,834][INFO ][cluster.service  ] [Lorna Dane] new_master [Lorna Dane][Hp5
p0uVuT1enV-PYkI-etQ][inet[/192.168.101.19:9300]], reason: zen-disco-join (elected_as_master)
[2014-02-04 09:36:57,877][INFO ][discovery        ] [Lorna Dane] elasticsearch/Hp5p0uVuT1enV
-PYkI-etQ
[2014-02-04 09:36:57,903][INFO ][http             ] [Lorna Dane] bound_address {inet[/0:0:0:
0:0:0:0:9200]}, publish_address {inet[/192.168.101.19:9200]}
[2014-02-04 09:36:57,904][INFO ][node                ] [Lorna Dane] started
[2014-02-04 09:36:57,939][INFO ][gateway          ] [Lorna Dane] recovered [0] indices into
cluster_state

```

You can see in the output that Elasticsearch started, that it assigned the node a random name, and that the node started a cluster and elected itself as master. The node is publishing to HTTP port 9200 (default). We use this port to interact with the cluster. Now you can

```
$ curl localhost:9200
```

and you get some data about your node



```

/Users/dennis/Projects/Blog/elasticsearch/curl.json — (80 x 20)
{
  "ok" : true,
  "status" : 200,
  "name" : "Cardiac",
  "version" : {
    "number" : "0.90.11",
    "build_hash" : "11da1bacf39cec400fd97581668acb2c5450516c",
    "build_timestamp" : "2014-02-03T15:27:39Z",
    "build_snapshot" : false,
    "lucene_version" : "4.6"
  },
  "tagline" : "You Know, for Search"
}
-:*** curl.json All (15,0) (JSON Fill)

```

Elasticsearch has a JSON based REST API. Administrative operations, indexing and searching, everything is done with HTTP and JSON. I use cURL for more concise examples. It may be more convenient for you to use any

graphical HTTP client. There is also a number of browser extensions and plugins. If you use Google Chrome then I recommend [Sense](#) plugin for Chrome, a JSON aware developer console to Elasticsearch.

Lets start up another node and give it a name *NODE_2* as parameter

```
$ bin/elasticsearch -f -Des.node.name=NODE_2
```

```
Ben:elasticsearch-0.90.11 dennis$ bin/elasticsearch -f -Des.node.name=NODE_2
[2014-02-04 09:43:00,262][INFO ][node                ] [NODE_2] version[0.90.11], pid[7256], bu
ild[11da1ba/2014-02-03T15:27:39Z]
[2014-02-04 09:43:00,263][INFO ][node                ] [NODE_2] initializing ...
[2014-02-04 09:43:00,273][INFO ][plugins            ] [NODE_2] loaded [], sites []
[2014-02-04 09:43:04,021][INFO ][node                ] [NODE_2] initialized
[2014-02-04 09:43:04,021][INFO ][node                ] [NODE_2] starting ...
[2014-02-04 09:43:04,276][INFO ][transport        ] [NODE_2] bound_address {inet[/0:0:0:0:0:
0:0:0:9301]}, publish_address {inet[/192.168.101.19:9301]}
[2014-02-04 09:43:07,440][INFO ][cluster.service    ] [NODE_2] detected_master [Volpan][0]s-W6
asQW6i7BxjShMHUg[inet[/192.168.101.19:9300]], added {[Volpan][0]s-W6asQW6i7BxjShMHUg[inet[/192.168
.101.19:9300]],}, reason: zen-disco-receive(from master [[Volpan][0]s-W6asQW6i7BxjShMHUg[inet[/192.
168.101.19:9300]])
[2014-02-04 09:43:07,479][INFO ][discovery          ] [NODE_2] elasticsearch/Ap3tCw1jRz2N1z9IX
OXFsQ
[2014-02-04 09:43:07,490][INFO ][http              ] [NODE_2] bound_address {inet[/0:0:0:0:0:
0:0:0:9201]}, publish_address {inet[/192.168.101.19:9201]}
[2014-02-04 09:43:07,491][INFO ][node                ] [NODE_2] started
█
```

You can see that *NODE_2* started. The node detected the other node as master and joined the cluster. The new node publishes to HTTP port 9201. You can talk to 9201 as well, as each node behaves the same. Any node starting up will join this cluster if, and only if it shares the cluster name. In our case as we haven't defined a cluster name a default setting is in place.

You now have a Elasticsearch cluster running with two nodes.

Install Head plugin

This is an optional step. The Head plugin or [elasticsearch-head](#) is a web front end for browsing and interacting with an Elasticsearch cluster. For more details on available plugins refer to the [plugin guide](#).

To install it run

```
$ bin/plugin -install mobz/elasticsearch-head
```

Then open http://localhost:9200/_plugin/head/ in your browser

Indexing

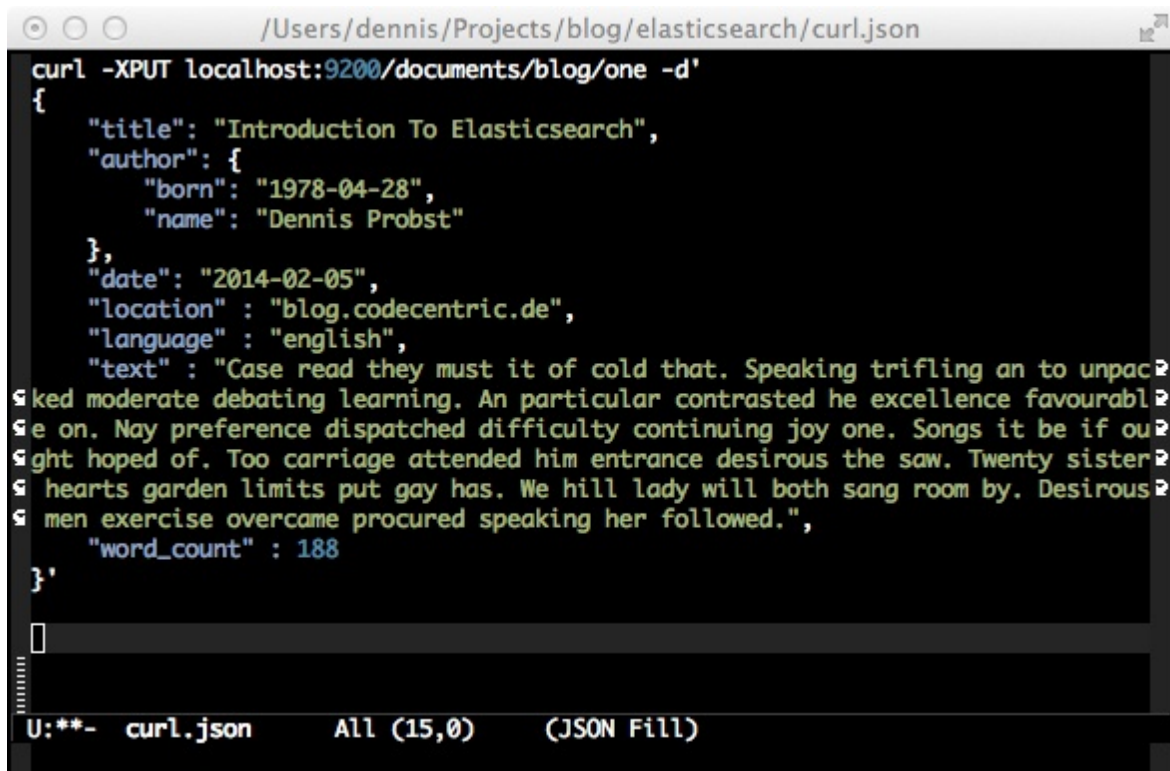
A Search engine is something like a database with a difference in how data is stored. The structure is loosely similar to that of a conventional relational database like MySQL or Postgres for example.

- **Elasticsearch - Database**
- Index - Database
- Type - Table
- Document - Row
- Field - Column

Elasticsearch creates an inverted index. http://en.wikipedia.org/wiki/Inverted_index

Everything is indexed in this data structure. It allows to quickly find all the documents that contain a particular word. Much in the way of an index at the back of a book.

Let's index something. We send off a HTTP PUT with the URL made up of the *index* name, *type* name and *ID* and in the HTTP payload we supply a JSON document with the fields and values. Notice the field *author* has another JSON object nested.



```
/Users/dennis/Projects/blog/elasticsearch/curl.json
curl -XPUT localhost:9200/documents/blog/one -d'
{
  "title": "Introduction To Elasticsearch",
  "author": {
    "born": "1978-04-28",
    "name": "Dennis Probst"
  },
  "date": "2014-02-05",
  "location": "blog.codecentric.de",
  "language": "english",
  "text": "Case read they must it of cold that. Speaking trifling an to unpac
ked moderate debating learning. An particular contrasted he excellence favourabl
e on. Nay preference dispatched difficulty continuing joy one. Songs it be if ou
ght hoped of. Too carriage attended him entrance desirous the saw. Twenty sister
hearts garden limits put gay has. We hill lady will both sang room by. Desirous
men exercise overcame procured speaking her followed.",
  "word_count": 188
}'
[]
U:**- curl.json All (15,0) (JSON Fill)
```

Indexing in Elasticsearch corresponds to *create* and *update* in CRUD. If we try to index a document with an ID that already exists it is overwritten. Index and type are required while the ID is optional. If we do not specify an ID, then Elasticsearch will generate one.

And we get a response that verifies that the operation was successful

```
{
  "ok": true,
  "_index": "documents",
  "_type": "blog",
  "_id": "one",
  "_version": 1
}
```

We get the name of the index, a type, and the ID. We also get a version, which is not a historical version. The versioning is used for optimistic concurrency control and is always incremented with any changes. The data we have supplied was all Elasticsearch needed. Elasticsearch automatically created the index for us.

Mapping

Elasticsearch is schema less. Elasticsearch is using mappings, which is basically a schema, but makes working with it much easier.

To see the mapping of our indexed blog

```
$ curl localhost:9200/documents/_mapping
```



```
{
  "documents": {
    "blog": {
      "properties": {
        "author": {
          "properties": {
            "born": {
              "type": "date",
              "format": "dateOptionalTime"
            },
            "name": {
              "type": "string"
            }
          }
        },
        "date": {
          "type": "date",
          "format": "dateOptionalTime"
        },
        "language": {
          "type": "string"
        },
        "location": {
          "type": "string"
        },
        "text": {
          "type": "string"
        },
        "title": {
          "type": "string"
        },
        "word_count": {
          "type": "long"
        }
      }
    }
  }
}
```

U:**~ curl.json Top (21,0) (JSON Fill)

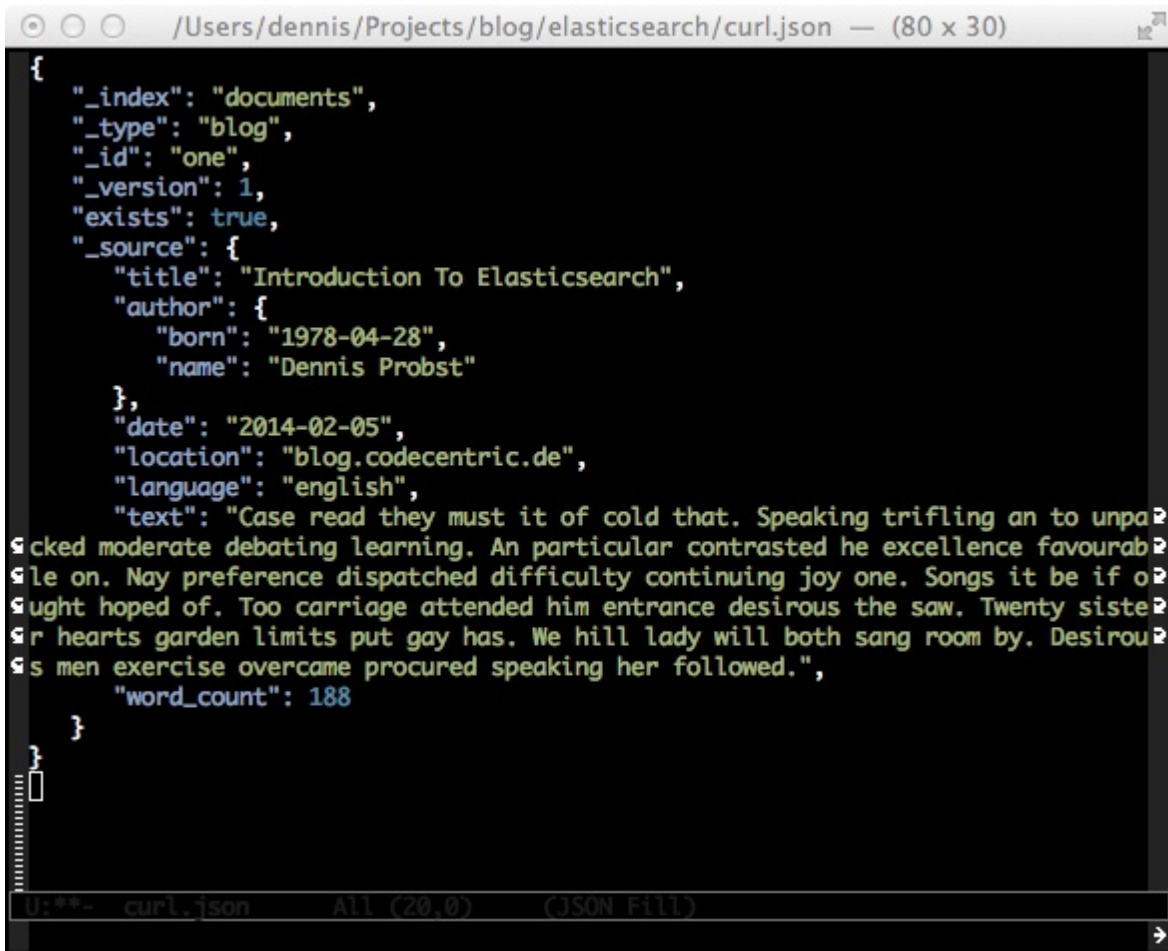
Here you get the mapping for the index *documents*. There is the type *blog* and a list of all properties from our blog. Elasticsearch automatically predicts the data types. If Elasticsearch doesn't predict the right type for your field, then you can supply a mapping when you create an index.

Getting data

To get a set of data, we set off a HTTP GET with the index name, type and the ID.

```
$ curl -XGET "http://localhost:9200/documents/blog/one"
```

The response looks like this



```

{
  "_index": "documents",
  "_type": "blog",
  "_id": "one",
  "_version": 1,
  "exists": true,
  "_source": {
    "title": "Introduction To Elasticsearch",
    "author": {
      "born": "1978-04-28",
      "name": "Dennis Probst"
    },
    "date": "2014-02-05",
    "location": "blog.codecentric.de",
    "language": "english",
    "text": "Case read they must it of cold that. Speaking trifling an to unpac
sacked moderate debating learning. An particular contrasted he excellence favourab
le on. Nay preference dispatched difficulty continuing joy one. Songs it be if o
ught hoped of. Too carriage attended him entrance desirous the saw. Twenty siste
r hearts garden limits put gay has. We hill lady will both sang room by. Desirou
s men exercise overcame procured speaking her followed.",
    "word_count": 188
  }
}

```

We get metadata information and the *source* field containing the JSON that we have indexed.

Searching

A document needs to be indexed before you can search for it. Elasticsearch refreshes every second by default.

Let's search for our data set

```
$ curl -XGET "http://localhost:9200/documents/blog/_search?q=_id:one"
```

```

{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 1,
    "hits": [
      {
        "_index": "documents",
        "_type": "blog",
        "_id": "one",
        "_score": 1,
        "_source": {
          "title": "Introduction To Elasticsearch",
          "author": {
            "born": "1978-04-28",
            "name": "Dennis Probst"
          },
          "date": "2014-02-05",
          "location": "blog.codecentric.de",
          "language": "english",
          "text": "Case read they must it of cold that. Speaking trifling a
          n to unpacked moderate debating learning. An particular contrasted he excellence
          favourable on. Nay preference dispatched difficulty continuing joy one. Songs i
          t be if ought hoped of. Too carriage attended him entrance desirous the saw. Twe
          nty sister hearts garden limits put gay has. We hill lady will both sang room by
          . Desirous men exercise overcame procured speaking her followed.",
          "word_count": 188
        }
      }
    ]
  }
}

```

curl.json All (33,1) (JSON Fill)

This is a query on the ID *one*. It is using the search facilities, but as we are looking for the ID, this can only result into one or zero documents.

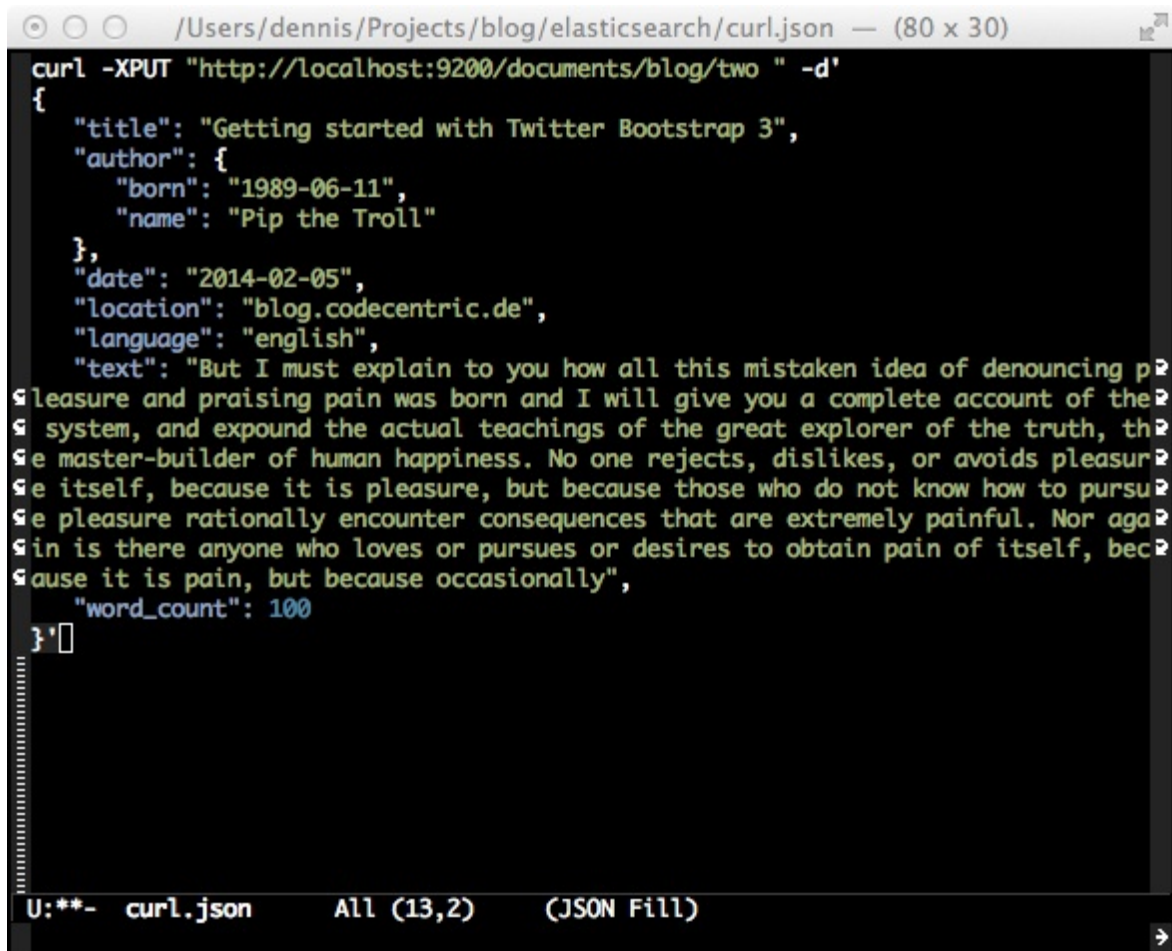
Lucene under the hood

Elasticsearch is built on top of Lucene, a very old Java library that is proven, tested and best of its kind in open source search software. Everything related to indexing and searching text is implemented in Lucene.

Elasticsearch builds an infrastructure around Lucene. While Lucene is a great tool, it can be cumbersome to use it directly and Lucene doesn't provide any facilities to scale past a single node. Elasticsearch provides an easier more intuitive API and the infrastructure and operational tools for simple scalability across multiple nodes. The REST API also allows interoperability with non-Java languages.

A *shard* in Elasticsearch refers to a Lucene index. Elasticsearch by default uses five shards for each Elasticsearch index. A document is stored in one shard. Elasticsearch supports replica shards. One replica is configured by default.

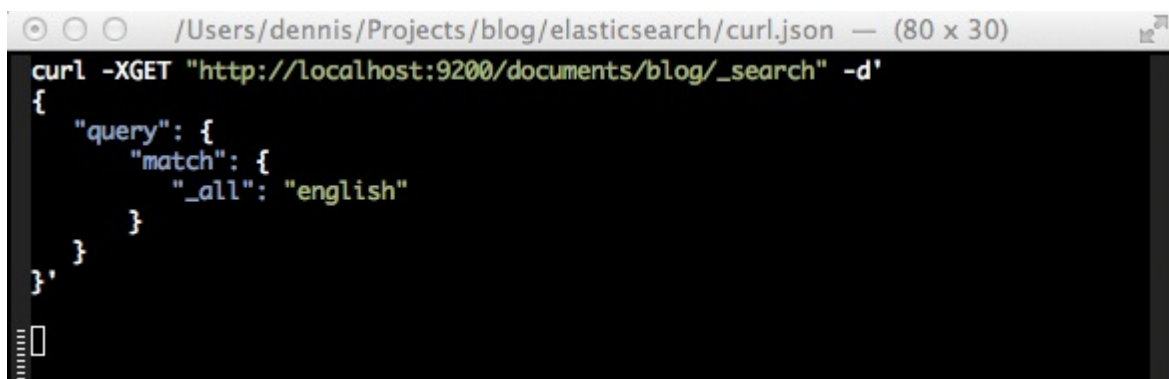
Let's put in another document



```
/Users/dennis/Projects/blog/elasticsearch/curl.json — (80 x 30)
curl -XPUT "http://localhost:9200/documents/blog/two" -d'
{
  "title": "Getting started with Twitter Bootstrap 3",
  "author": {
    "born": "1989-06-11",
    "name": "Pip the Troll"
  },
  "date": "2014-02-05",
  "location": "blog.codecentric.de",
  "language": "english",
  "text": "But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. No one rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because those who do not know how to pursue pleasure rationally encounter consequences that are extremely painful. Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but because occasionally",
  "word_count": 100
}'
```

U:**~ curl.json All (13,2) (JSON Fill)

Now let's search for the term *english*



```
/Users/dennis/Projects/blog/elasticsearch/curl.json — (80 x 30)
curl -XGET "http://localhost:9200/documents/blog/_search" -d'
{
  "query": {
    "match": {
      "_all": "english"
    }
  }
}'
```

we get two hits

```

/Users/dennis/Projects/Blog/elasticsearch/curl.json — (80 x 49)
{
  "took": 25,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 2,
    "max_score": 0.033562027,
    "hits": [
      {
        "_index": "documents",
        "_type": "blog",
        "_id": "two",
        "_score": 0.033562027,
        "_source": {
          "title": "Getting started with Twitter Bootstrap 3",
          "author": {
            "born": "1989-06-11",
            "name": "Pip the Troll"
          },
          "date": "2014-02-05",
          "location": "blog.codecentric.de",
          "language": "english",
          "text": "But I must explain to you how all this mistaken idea of
denouncing pleasure and praising pain was born and I will give you a complete ac-
count of the system, and expound the actual teachings of the great explorer of t-
he truth, the master-builder of human happiness. No one rejects, dislikes, or av-
oids pleasure itself, because it is pleasure, but because those who do not know
how to pursue pleasure rationally encounter consequences that are extremely pain-
ful. Nor again is there anyone who loves or pursues or desires to obtain pain of
itself, because it is pain, but because occasionally",
          "word_count": 100
        }
      },
      {
        "_index": "documents",
        "_type": "blog",
        "_id": "one",
        "_score": 0.033562027,
        "_source": {
          "title": "Introduction To Elasticsearch",
          "author": {
            "born": "1978-04-28",
            "name": "Dennis Probst"
          }
        }
      }
    ]
  }
}
-:***- curl.json      Top (40,1)      (JSON Fill)

```

We can also search on a specific field. Nested fields are addressed with a point separator like in the next example.



```
/Users/dennis/Projects/Blog/elasticsearch/curl.json — (80 x 15)
curl -XPOST "http://localhost:9200/documents/blog/_search" -d'
{
  "query": {
    "prefix": {
      "author.name": "pip"
    }
  }
}'
-:***-  curl.json  All (8,2)  (JSON Fill)
```

We get a result that matches the author name *Pip the Troll*

```

/Users/dennis/Projects/Blog/elasticsearch/curl.json — (80 x 46)
{
  "took": 615,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 1,
    "hits": [
      {
        "_index": "documents",
        "_type": "blog",
        "_id": "two",
        "_score": 1,
        "_source": {
          "title": "Getting started with Twitter Bootstrap 3",
          "author": {
            "born": "1989-06-11",
            "name": "Pip the Troll"
          },
          "date": "2014-02-05",
          "location": "blog.codecentric.de",
          "language": "english",
          "text": "But I must explain to you how all this mistaken idea of
denouncing pleasure and praising pain was born and I will give you a complete ac
count of the system, and expound the actual teachings of the great explorer of t
he truth, the master-builder of human happiness. No one rejects, dislikes, or av
oids pleasure itself, because it is pleasure, but because those who do not know
how to pursue pleasure rationally encounter consequences that are extremely pain
ful. Nor again is there anyone who loves or pursues or desires to obtain pain of
itself, because it is pain, but because occasionally",
          "word_count": 100
        }
      }
    ]
  }
}
-:***-  curl.json  All (33,1)  (JSON Fill)

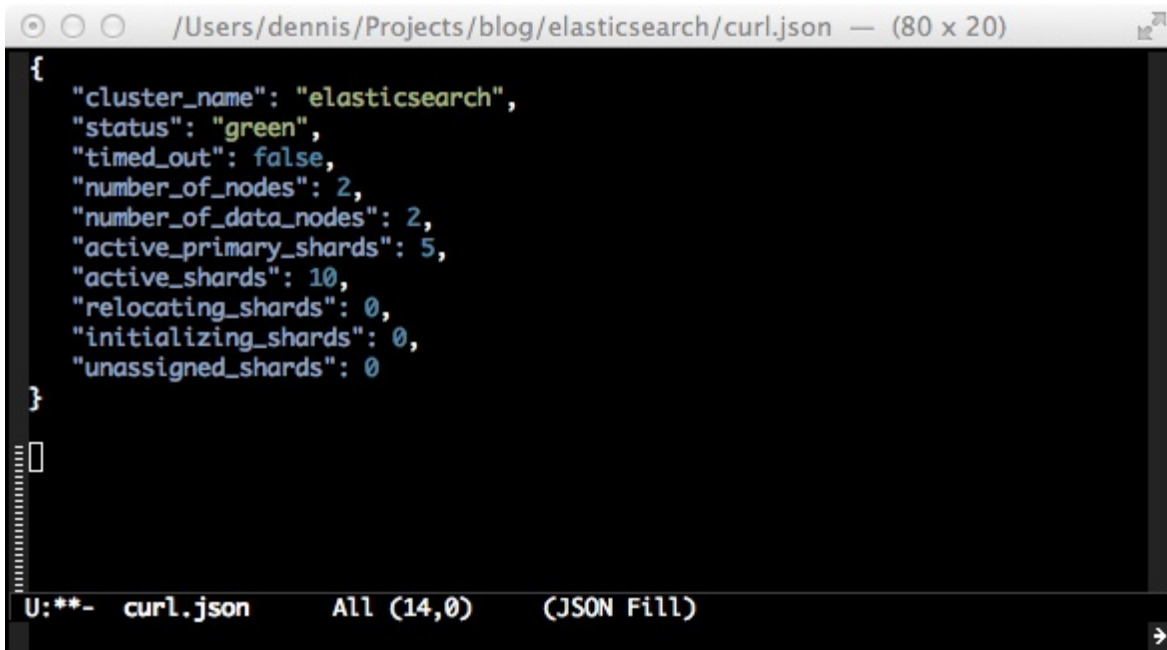
```

In the last example we used a prefix query. For more types of queries refer to the [elasticsearch.org guide](http://elasticsearch.org/guide). We are not providing the full value of the field. This is the gist of a search engine. We don't need to know exactly what we are searching for. We can provide what we know and get results on what might be true or not in contrary to what *must be true*. We can find word stems, synonyms, misspellings, and we can even provide autocompletion.

Clustering

You can get some information about your cluster with

```
$ curl -XGET "http://localhost:9200/_cluster/health"
```



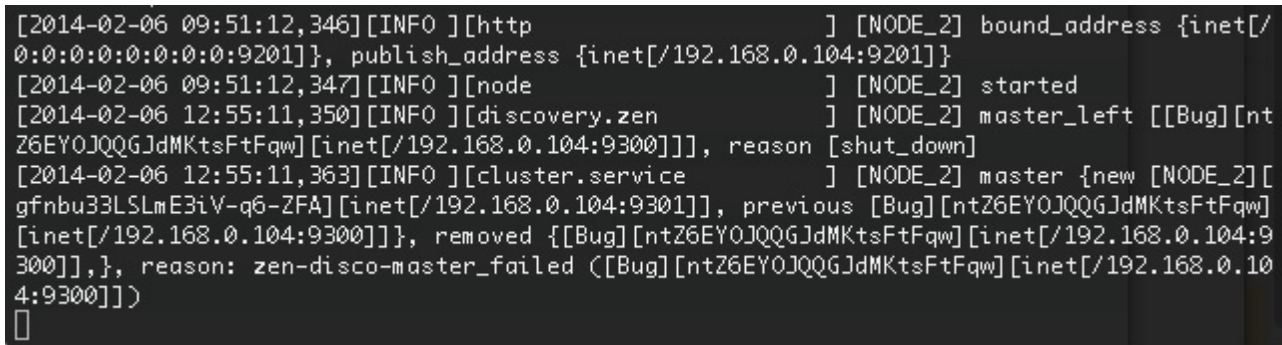
```

/Users/dennis/Projects/blog/elasticsearch/curl.json — (80 x 20)
{
  "cluster_name": "elasticsearch",
  "status": "green",
  "timed_out": false,
  "number_of_nodes": 2,
  "number_of_data_nodes": 2,
  "active_primary_shards": 5,
  "active_shards": 10,
  "relocating_shards": 0,
  "initializing_shards": 0,
  "unassigned_shards": 0
}
U:**~ curl.json All (14,0) (JSON Fill)

```

Alternatively, if you have installed the head plugin you can open http://localhost:9201/_plugin/head/. We have a status, which is green. We have 5 active primary shards and 10 active shards, because we're running with two nodes. Our indexed documents are available on each node.

Lets shut down our master. Go to your console of your master node and press CTRL-C then look at the console output of *NODE_2*.



```

[2014-02-06 09:51:12,346][INFO ][http                ] [NODE_2] bound_address {inet[/
0:0:0:0:0:0:0:0:9201]}, publish_address {inet[/192.168.0.104:9201]}
[2014-02-06 09:51:12,347][INFO ][node                ] [NODE_2] started
[2014-02-06 12:55:11,350][INFO ][discovery.zen      ] [NODE_2] master_left [[Bug][nt
Z6EY0JQQGJdMKtsFtFqw][inet[/192.168.0.104:9300]]], reason [shut_down]
[2014-02-06 12:55:11,363][INFO ][cluster.service    ] [NODE_2] master {new [NODE_2][
gfnbu33LSLmE3iV-q6-ZFA][inet[/192.168.0.104:9301]], previous [Bug][ntZ6EY0JQQGJdMKtsFtFqw]
[inet[/192.168.0.104:9300]]}, removed {[Bug][ntZ6EY0JQQGJdMKtsFtFqw][inet[/192.168.0.104:9
300]]}, reason: zen-disco-master_failed ([Bug][ntZ6EY0JQQGJdMKtsFtFqw][inet[/192.168.0.10
4:9300]])

```

You can see that *NODE_2* noticed that the master has left the cluster and elected itself as new master. Check the status again

```
$ curl -XGET "http://localhost:9201/_cluster/health"
```

Make sure to use port 9201 of *NODE_2* not 9200.



```
/Users/dennis/Projects/Blog/elasticsearch/curl.json — (80 x 15)

{"cluster_name": "elasticsearch",
 "status": "yellow",
 "timed_out": false,
 "number_of_nodes": 1,
 "number_of_data_nodes": 1,
 "active_primary_shards": 5,
 "active_shards": 5,
 "relocating_shards": 0,
 "initializing_shards": 0,
 "unassigned_shards": 5}

--:**-- curl.json All (12,1) (JSON Fill)
```

You can see that the cluster status is now yellow, because one of our two nodes is unassigned. The search functionality of the cluster is still working, though. If you set off our search requests from earlier again, but this time against port 9201 from *NODE_2*, you still get the search results, because everything we have indexed is available on every node.

If you start our previous master back up and check on the cluster status it'll be back in status *green*.

Facets

At some point you will be interested in information about the data you have indexed. In our case we might be interested in what is the average number of words over all indexed blog articles. Elasticsearch has a feature called facets that provides aggregated statistics about a query. This is a core part of Elasticsearch and is part of the search API. Facets are always bound to a query and provide aggregate statistics alongside the query results. Facets are highly configurable and can return complex groupings of nested filters, spans of amounts or spans of time, even full Elasticsearch queries can be nested inside a Facet. In Elasticsearch 1.0 this feature will be called Aggregations and is supposed to have more features and be more composable.

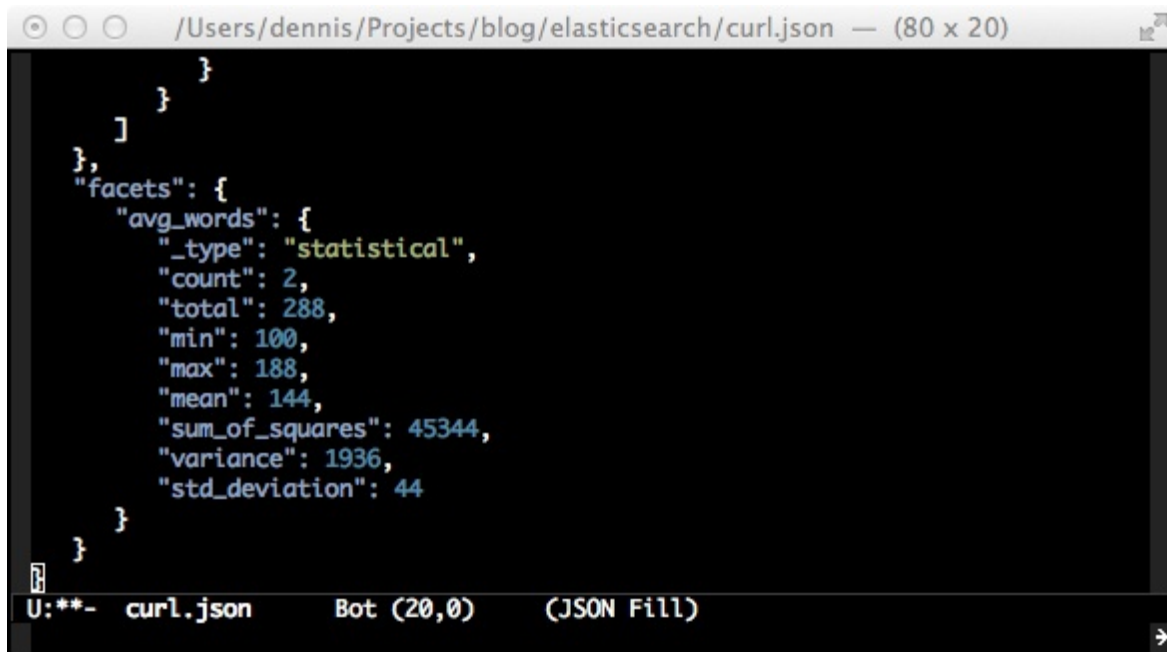


```
/Users/dennis/Projects/Blog/elasticsearch/curl.json — (80 x 18)

curl -XPOST "http://localhost:9201/documents/blog/_search" -d'
{
  "facets": {
    "avg_words": {
      "statistical": {
        "field": "word_count"
      }
    }
  },
  "query": {
    "match_all": {}
  }
}
}'

--:**-- curl.json All (13,2) (JSON Fill)
```

Here we define a facet together with a match all query. The facet is a predefined statistical facet for number fields in this case our *word_count* field.



```

{
  "facets": {
    "avg_words": {
      "_type": "statistical",
      "count": 2,
      "total": 288,
      "min": 100,
      "max": 188,
      "mean": 144,
      "sum_of_squares": 45344,
      "variance": 1936,
      "std_deviation": 44
    }
  }
}

```

In the response we receive the query results first on top then the facet response with statistical numbers on our *word_count* field. There are further predefined facets you can choose from in order to get statistical information about your data and there is also [Kibana](#) that is a graphical front-end data analysis tool specifically tailored for Elasticsearch that became very popular.

That's it!

No of course that's not it. It's just all I wanted to show you in this 101. I've introduced you to quiet a few topics but we barely scratched the surface of what there is to discover about Elasticsearch.

The query API has much more to offer than we have covered for instance. There are many interesting types of queries and filters that can be used. To get the most out of natural language based searches and other complex types of data, you'll get in touch with analyzers. Analyzers are the tools to slice and dice words into stems to create an efficient search space for natural languages. The word stemming allows Elasticsearch to find linguistically similar words. Percolators are another very interesting topic. Percolators allow one to index queries and then send docs to Elasticsearch and find out which queries match the doc. So the entire operation turned around going in reverse direction. And there is even more.

I hope you found this post interesting and useful on your quest to discover this awesome peace of technology. Thank you for reading and stay in the loop for more posts to come on Elasticsearch. In the meanwhile I've put a list of links together, you can find them below, there is also a great interview with Github about Elasticsearch at scale.

Where to go from here

- Visit the elasticsearch.org guide
- You may follow the [elasticsearch.org](http://elasticsearch.org/blog) blog
- Tutorials can be found at [elasticsearch.org](http://elasticsearch.org/tutorials) tutorials
- Find Elasticsearch projects on [github](https://github.com)
- Article about [Elasticsearch Analyzers](#)
- Get [Elasticsearch running on EC2](#)
- To use Elasticsearch with Java go to the [elasticsearch.org](http://elasticsearch.org/java-api) Java API
- [Marvel](#) is a visual cluster health and metrics dashboard
- For graphical data analysis check out [Kibana](#)

To learn more about [Lucene](#) go to the [Lucene documentation](#) or visit the [Lucene wiki](#).

Inverview with Github about Elasticsearch at scale

<http://exploringelasticsearch.com/book/elasticsearch-at-scale-interviews/interview-with-the-github-elasticsearch-team.html>

Tweet

10

g+1

2



category: [Search](#) | [no comment](#)

This post is written on February 7, 2014 by [dennis.probst](#)

Weitere Beiträge aus dem [codecentric Blog](#)

Ähnliche Beiträge:

- [Elasticsearch Indexing Performance Cheatsheet](#)
- [Elasticsearch Monitoring and Management Plugins](#)
- [Test Automation for NoSQL Databases with NoSQL Unit and Travis-CI](#)
- [Behaviour Driven Development with Elasticsearch](#)
- [Installing a Hadoop Cluster with three Commands](#)

Aktuelle Beiträge:

- [Elasticsearch Indexing Performance Cheatsheet](#)
- [Elasticsearch Monitoring and Management Plugins](#)
- [Behaviour Driven Development with Elasticsearch](#)
- [Elasticsearch 101](#)

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes: <abbr title=""> <acronym title=""> <blockquote cite=""> <cite> <code> <del datetime=""> <i> <q cite=""> <strike>

Post Comment

-  [English](#)
-  [Deutsch](#)



© 2014 codecentric AG **codecentric AG** Kölner Landstr. 11 | 40591 Düsseldorf | Telefon: +49 (0) 211.99414-0 | Fax: +49 (0) 211.99414-44 | E-Mail: info@codecentric.de

