ELASTICSEARCH TUTORIAL FOR BEGINNERS: ELASTICSEARCH BASICS



ElasticSearch (ES) is a noSQL JSON (not only SQL JavaScript Object Notation) database. Its primary application is to store logs from applications, network devices, operating systems, etc. But it is suitable for the storage of any kind of JSON document. So, you could use it instead of, for example, MongoDB. Yet, MongoDB has native support for JavaScript, as we explained here, which you will find useful. ES does has not such a REPL (read-eval-print loop) command line interface, except for Curator, which can be used for admin functions.

(This article is part of our ElasticSearch Guide. Use the right-hand menu to navigate.)

What's included in this Guide

In this guide we will cover the most important ElasticSearch topics.

This includes:

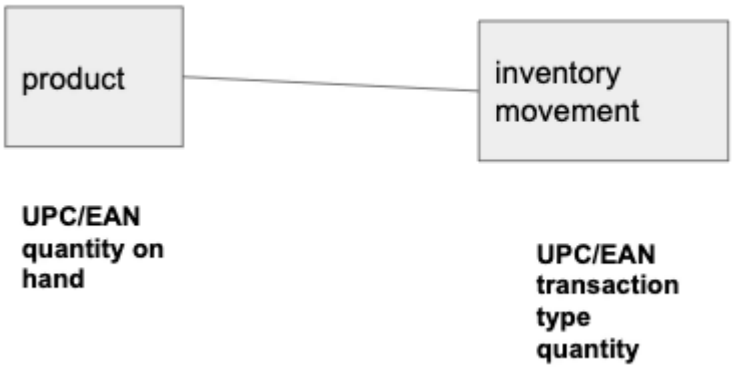
- How to set up an ElasticSearch Cluster, giving both versions 6.x and 7.x instructions, since version 7 is substantially different.
- How to use Kibana (i.e., the ES dashboard)
- The Lucent Query Language
- How JSON databases differ from traditional SQL databases, like Oracle
- Using Apache PIG with ES
- Using Apache Spark and Apache Spark Machine Learning with ES
- ElasticSearch Indexes

- Using Filebeat and Logstash to parse web server, router, and custom and off-the-shelf application logs.
- Nesting documents within documents
- System administration
- Adding clusters and re-indexing documents

JSON documents and noSQL

First it is necessary to understand what a JSON database is and does. Traditionally, databases store records in tables in columns. That is called a relational database (RDBMS). The most widely-known example of this is Oracle, invented in the 1970s from a paper written by IBM, a product that has made Larry Ellison quite a rich man and a product that gave him a monopoly for many years. (There are open source alternatives, like MySQL. But what do monopolies do? They buy up the competition. Oracle has bought MySQL.)

Data stored in an RDBMS is put together in a join operation and the data is normalized in most cases. As an example of this, consider an inventory system. You have products and then a table of inventory movements (sale, stock, loss, etc.). So, you would probably store these in two separate tables, like this:



The common elements upon which you join the two are the UPC/EAN numbers (Universal Product Code and and European Article Number.) Those are the barcodes scanned at the grocery store. The product keeps the inventory count. The inventory movement table keeps the sale, receipt, loss, and other transactions on hand.

JSON databases do not support, or rather encourage, join operations, although that operation can be forced. Why? Because joining two tables is an expensive operation, because it must scan all the records of one table to find match records in another. Of course, that operation can be sped up using indexes.

Instead of doing joins, the JSON database embeds documents inside each other, like shown below, where the transactions are inside the product document.

```
{
"product": "led lightbulbs 10w 6 pack",
"EAN": 978020137962,
"transactions": ,
"stock": 8000
}
```

LogStash

It would do not much good to dump logs in their native format into ElasticSearch since you could not query them in a standard way.

For example, if you operate a web server and dump the logs directly into ElasticSearch they would not be easy to read, since they would not be parsed into individual fields. What ElasticSearch does (using Logstash or Filebeat or both) is parse each line into JSON format. For example, an nginx web server log looks like this:

```
191.100.8.229 - -  "GET / HTTP/1.1" 200 3100 "-" "Mozilla/5.0 (Windows NT
10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103
Safari/537.36"

103.206.128.70 - -  "GET / HTTP/1.1" 200 3100 "-" "Mozilla/5.0 (Windows NT
10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116
Safari/537.36"
```

You turn that into JSON using the ElasticSearch tools Logstash, Grok, and Filebeat.

The Curl Interface

If you work with a JSON database then it's important to know its APIs (Application Programming Interface). Because it uses APIs, this means you can talk to it any with programming languages. And because there are SDKs (Software Developer Kit) too you can talk to it with specific programming languages. The difference is that APIs work over HTTP and HTTPS, meaning there is no need for a direct connection to the system over a port that is perhaps not open. Additionally, APIs are limited to the operations GET, PUT, POST, and DELETE.

For example, you can list indexes like this:

```
curl -XGET http://172.31.46.15:9200/_cat/indices?v


health status index                            uuid                   pri rep
docs.count docs.deleted store.size pri.store.size
green  open   .kibana_task_manager             k00ncRwwTJKTA41OF7QTfg   1   1
2            0     62.4kb         31.2kb
green  open   .monitoring-kibana-6-2019.05.14  eaNOm7yXT8m62DEqv1Zkow   1   1
251            0    420.7kb        210.3kb
green  open   .kibana_2                        cTq9L_2zT0yfDhmChb0njQ   1   1
616           77    716.6kb        358.3kb
green  open   .monitoring-es-6-2019.05.15      ZgNBg16SS4qDkzuMctzchg   1   1
```

And you can write to ES like this:

```
curl -XPUT --header 'Content-Type: application/json'
http://localhost:9200/samples/_doc/1 -d '{
   "school" : "Harvard"
}'
```

And then query that with:

```
curl -XGET --header 'Content-Type: application/json'
http://localhost:9200/samples/_search -d '{
      "query" : {
        "match" : { "school": "Harvard" }
    }
}'
```

And you can do all of that with the Kibana graphical interface as well.

Note: of the last example purists would say is not logical since the GET operation does not take body (i.e., the -d) parameters. However, ElasticSearch says it does, since the RFC spec for web operations says so.

So, buckle in as we take you through ElasticSearch. It's going to be a wild ride.