



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



Dipartimento
di Informatica

Text Analytics, Search and Personalization

ElasticSearch - Intro

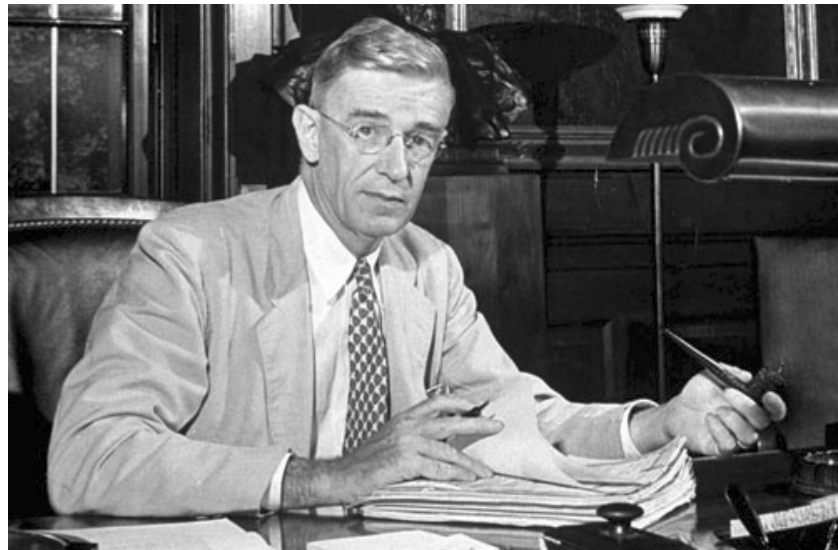
Docenti:

dott. Marco Polignano – marco.polignano@uniba.it



Introduction

Information Retrieval?



VANNEVAR BUSH

1945 - As We May Think

There is, of course, provision for consultation of the record by the usual scheme of indexing. If the user wishes to consult a certain book, he **taps its code on the keyboard**, and the title page of the book promptly appears before him, projected onto one of his viewing positions.

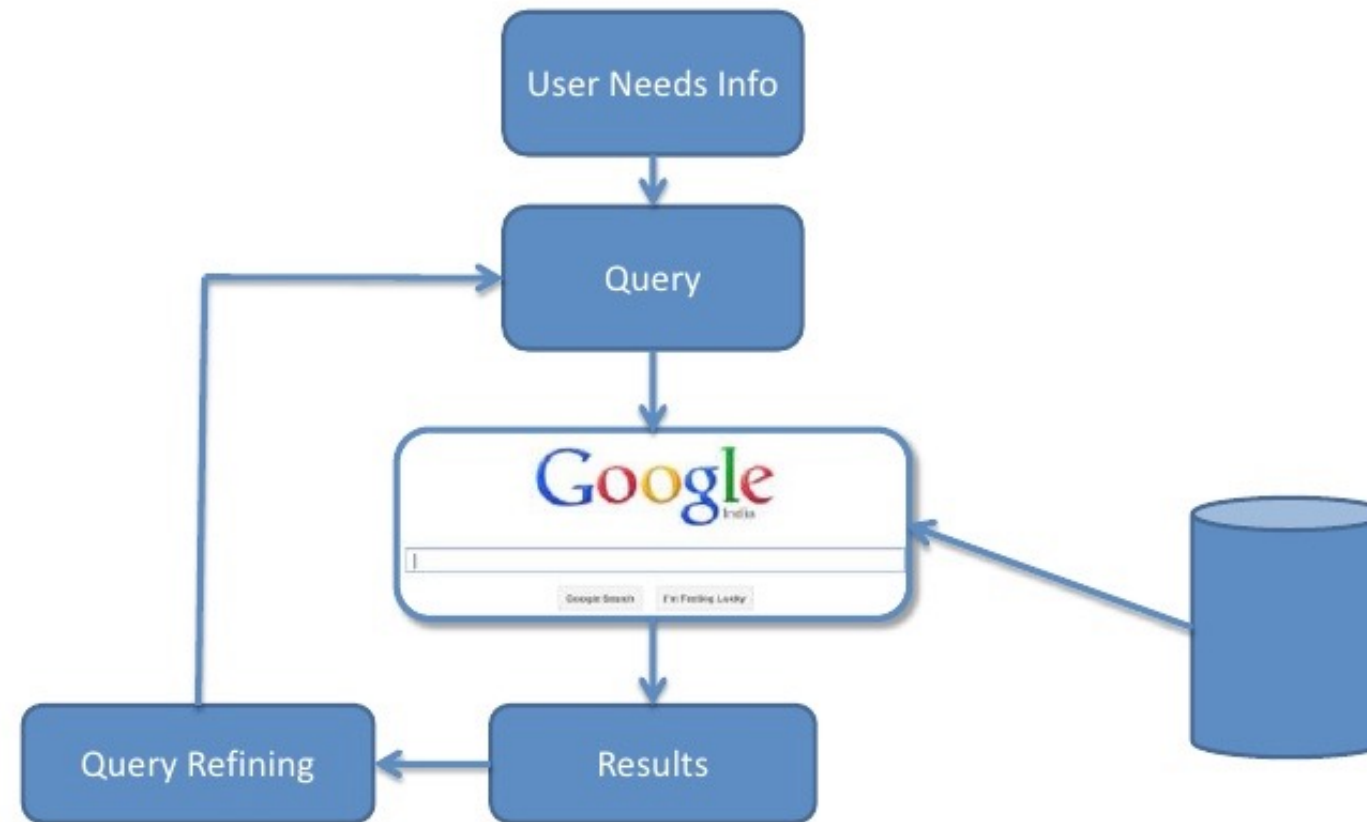
A special button transfers him immediately to the **first page of the index**. Any given book of his library can thus be called up and consulted with far greater facility than if it were taken from a shelf. As he has several projection positions, he can leave one item in position while he calls up another.

Introduction

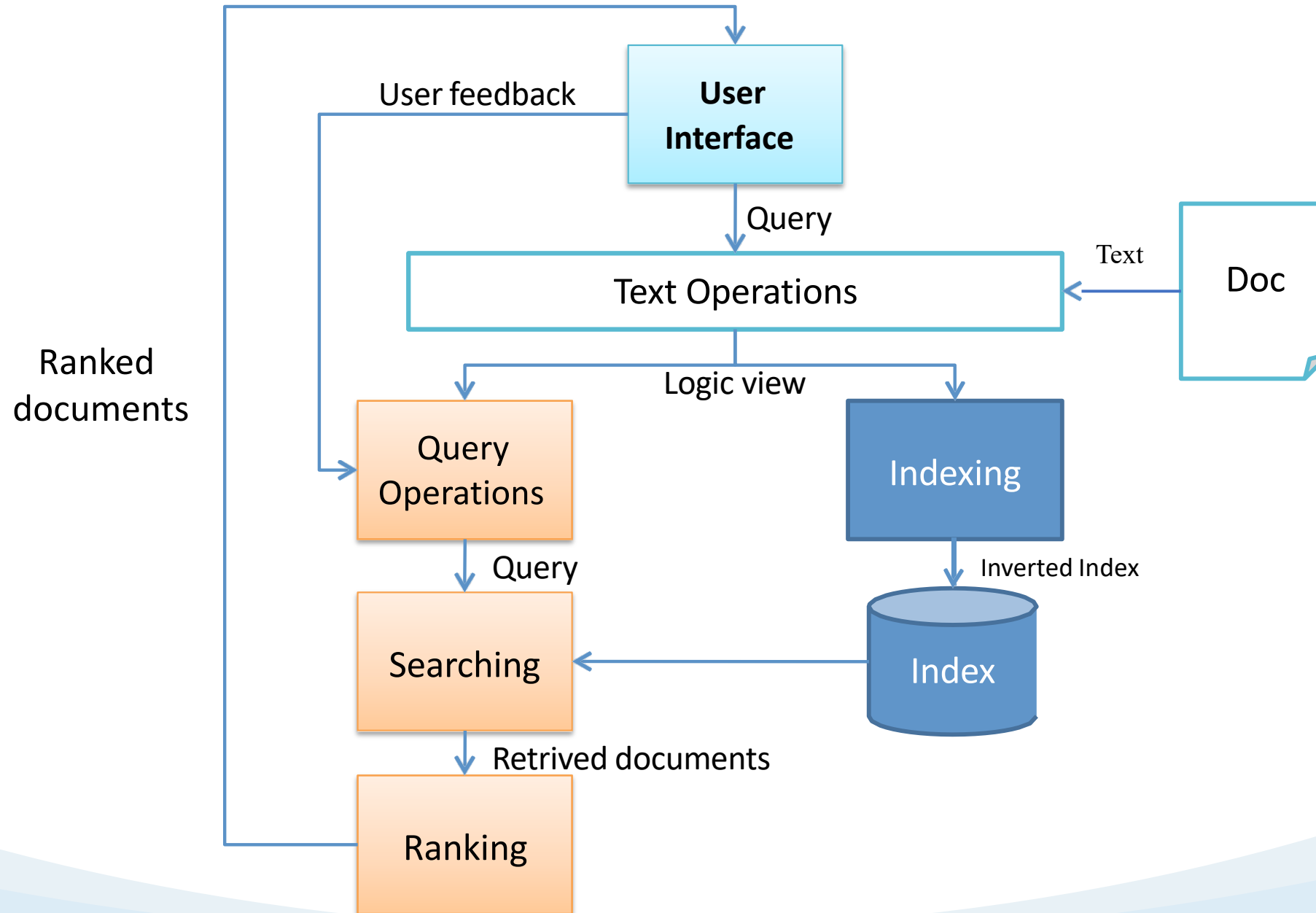


Introduction

Basic Search Model



Information Retrieval Process



Information Retrieval Framework



www.elastic.co

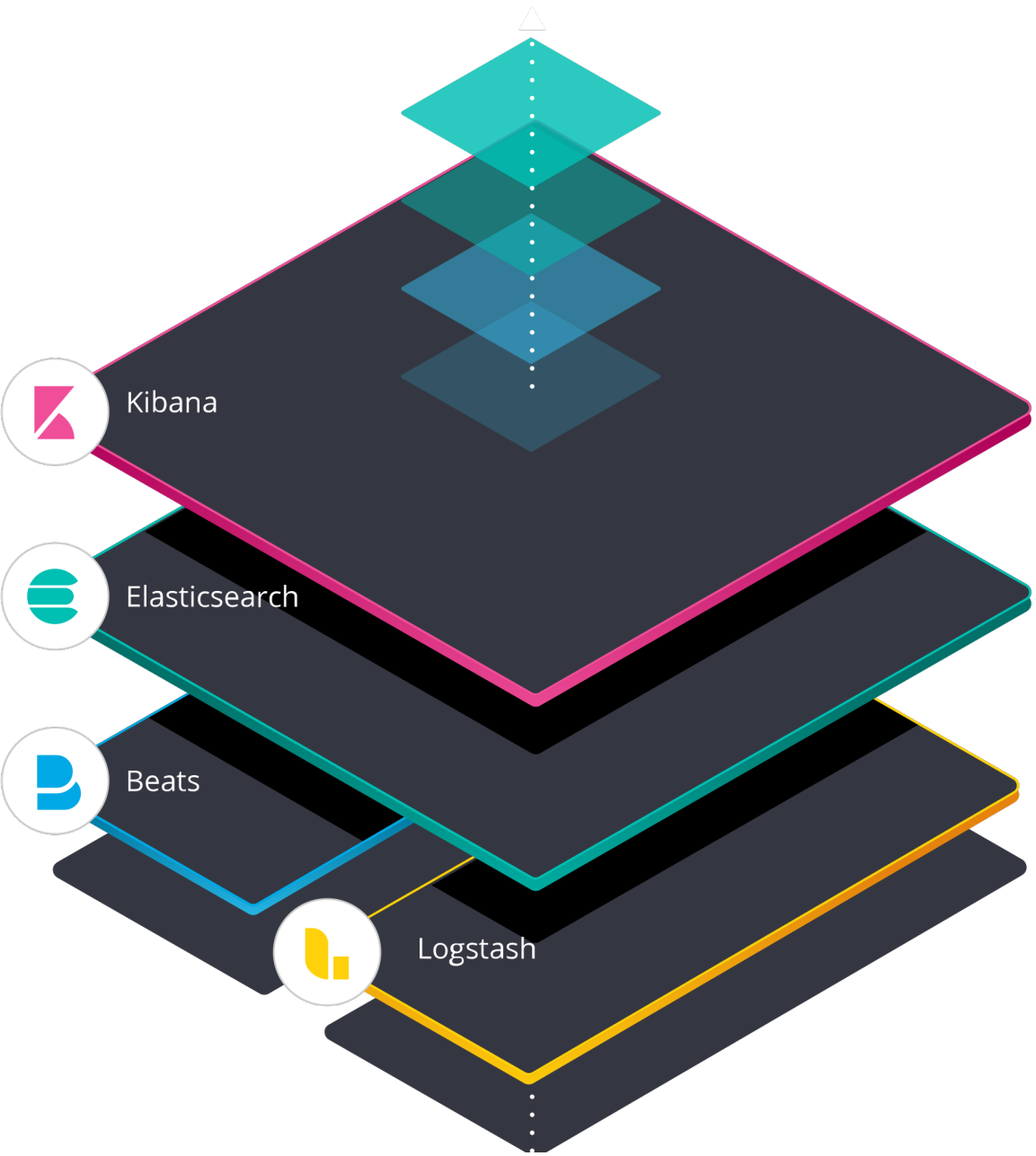
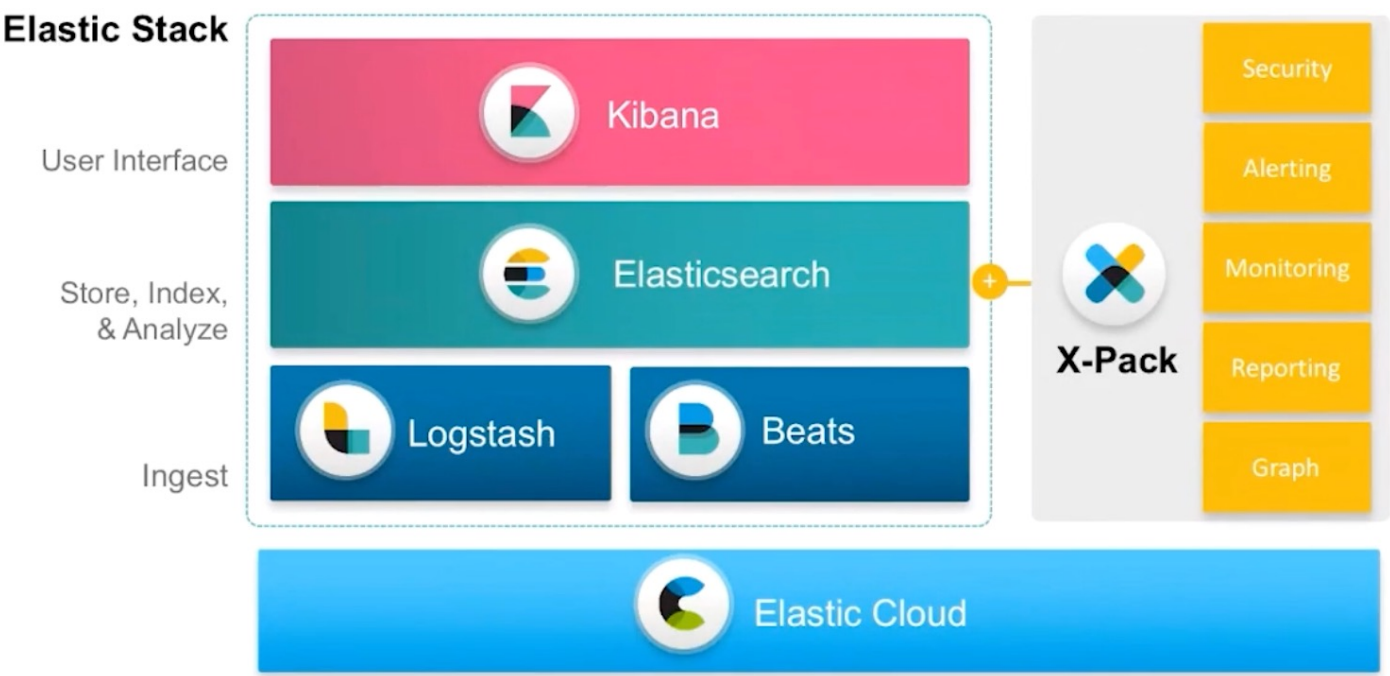
www.elastic.co/webinars/getting-started-elasticsearch

- **ElasticSearch** is a distributed, **JSON-based search and analytics engine**
- Build on top of **Apache Lucene**
 - Lucene is a most popular java-based full text indexing and retrieval library
- First public release version **v0.4 in February 2010**
- Developed in **Java**, so inherently cross-platform

Information Retrieval Framework

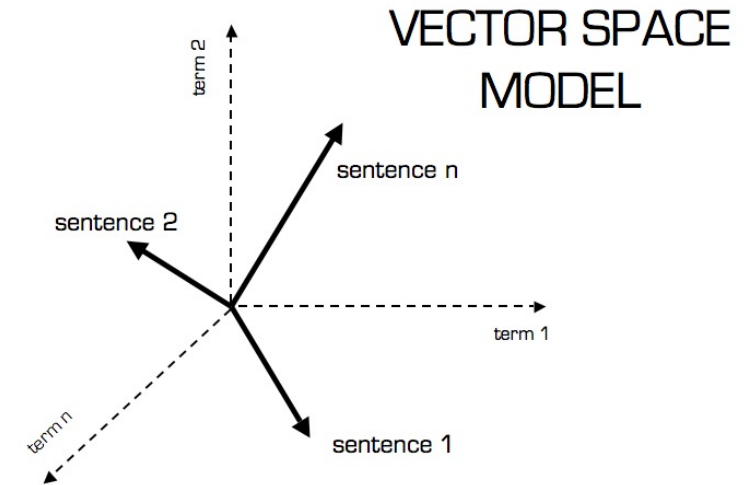


Introducing the Elastic Stack, X-Pack, and Cloud

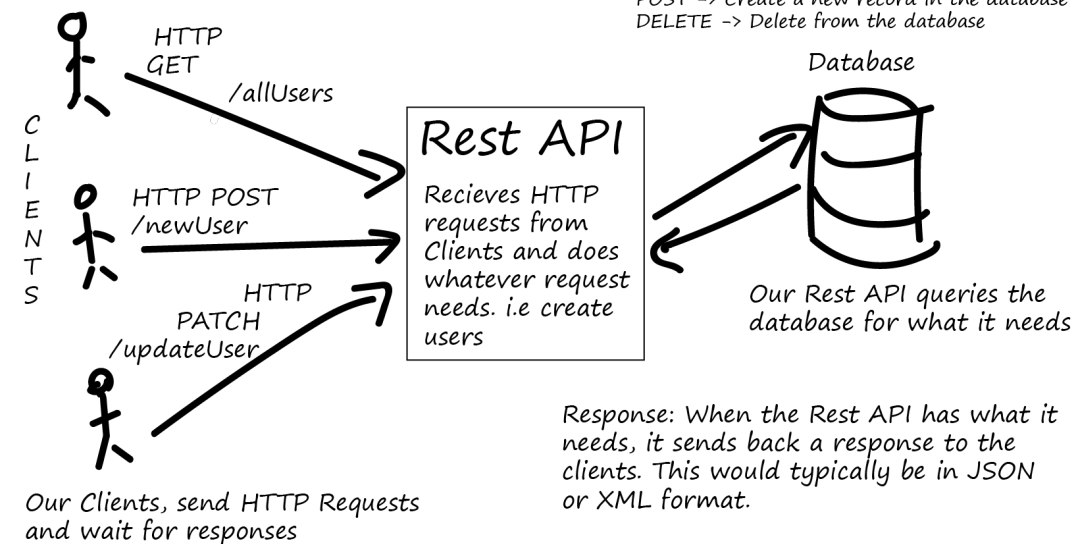


Why Elasticsearch?

- Easy to scale (**Distributed**)
- Everything is one JSON call away (**RESTful API**)
- Unleashed power of Lucene under the hood
- Multi-tenancy
- Support for advanced search features (Full Text)
- Configurable and **Extensible**
- **Document Oriented**



Rest API Basics



Let's Start!

➤ Cluster :

A cluster consists of **one or more nodes which share the same cluster name**. Each cluster has a **single master node** which is chosen automatically by the cluster and which can be replaced if the current master node fails.

➤ Node :

A node is **a running instance of elasticsearch** which belongs to a cluster. At startup, a node will use unicast to **discover an existing cluster** with the same cluster name and will try to join that cluster.

➤ Shard :

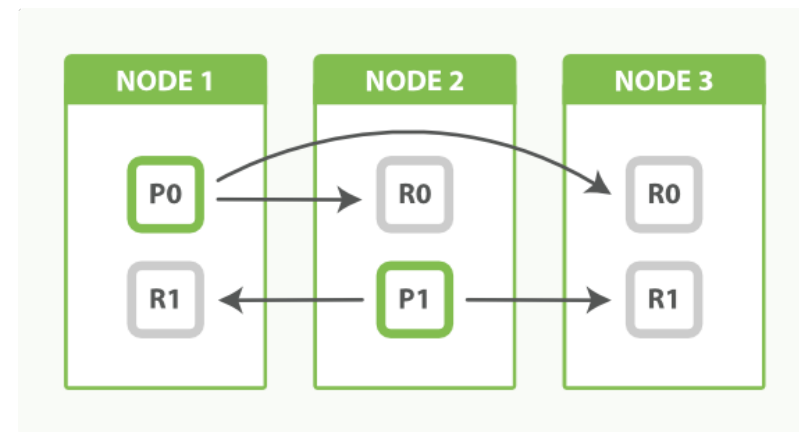
A shard is a **single Lucene instance**. It is a **low-level “worker”** unit which is managed automatically by elasticsearch. An index is a logical namespace which points to primary and replica shards.

Easy to Scale!

- It is built to **scale horizontally** out of the box.
- One server can **hold one or more parts of one or more indexes**, and whenever new nodes are introduced to the cluster they **are just being added to the party**.
- Every such index, or part of it, is called a shard, and Elasticsearch shards can be moved around the cluster very easily.

```
PUT /my_index
{
  "settings": {
    "number_of_shards": 3,
    "number_of_replicas": 2
  }
}
```

CLUSTER



Basic Concepts 2

➤ Index :

An index is like a ‘**database**’ in a relational database. It has a mapping which defines multiple types.

An index maps to one or more primary shards and can have *zero or more replica shards*

➤ Document :

A document is a JSON **document which is stored in elasticsearch**. It is *like a row in a table* in a relational database. Each document is stored in an index and has an id.

A document is a JSON object which contains zero or more fields, or key-value pairs.

The original JSON document that is indexed will be stored in the **_source field**, which is returned by default when getting or searching for a document.

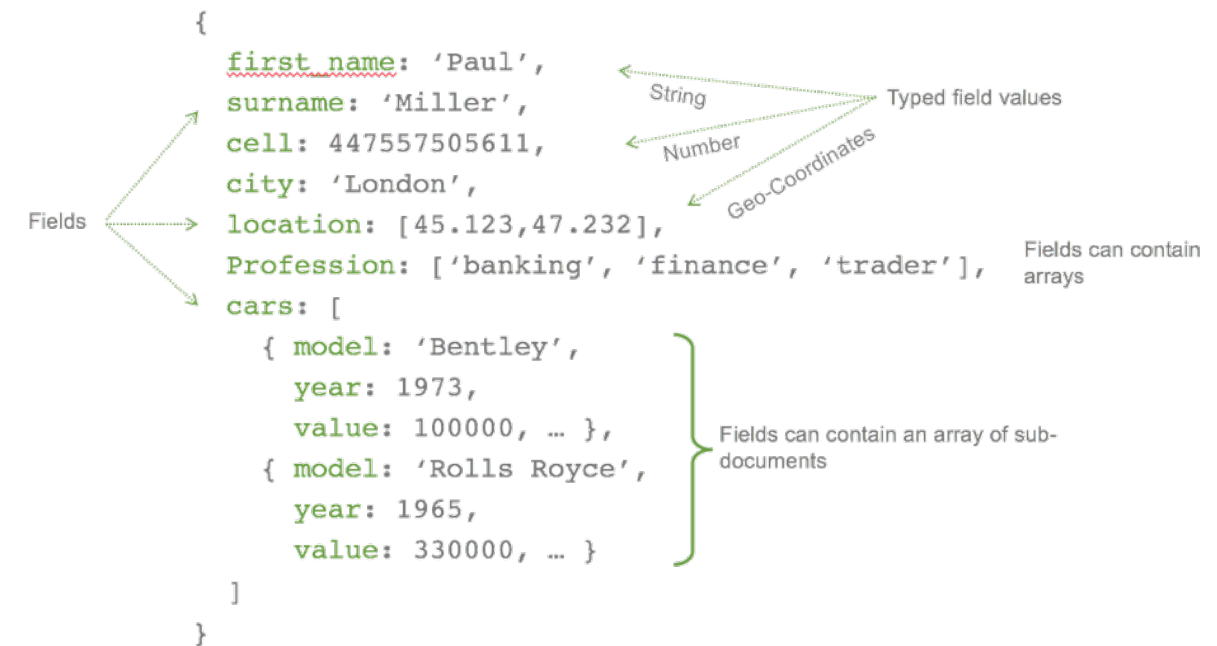
Basic Concepts 2

➤ Field:

A **document** contains a **list of fields**, or **key-value pairs**. The value can be a simple (scalar) value (eg a string, integer, date), or a nested structure like an array or an object.

The mapping for each field has a field **'type'** which indicates the type of data that can be stored in that field, eg integer, string, object.

This mapping allows you to define (amongst other things) ***how the value for a field should be analyzed.***



DOCUMENT

A Document can be described as:

- A collection of **Fields** which give structure to the document
- Each Field is ***added to the document***
- A Field has three parts
 - **Name**
 - **Type**
 - **Value**

Field TYPES

TextField: String indexed for full-text search

Keyword: String indexed as a single term!

Integer: int indexed for exact/range queries

Long: long indexed for exact/range queries

Float: float indexed for exact/range queries

Double: double indexed for exact/range queries

Geopoints: for storing latitude and longitude data

Date: time and date field

<https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html>

JSON?!

JSON (JavaScript Object Notation) is a **lightweight data-interchange format**.

- *It is **easy for humans** to read and write*
- *It is **easy for machines** to parse and be generated*

```
{  
  "person":  
    { "name" : "John", "age" : 31, "city" : "New York" }  
}
```

Let's Start Again!

- › Oracle Java SDK **11** (preferably not OpenJDK)
- › *JAVA IDE: (optional)*
 - › *NetBeans, Eclipse, IntelliJ, ...*
- › *Python 3.x and Jupyter Notebook*
- › ElasticSearch: <https://www.elastic.co/downloads/elasticsearch>
- › Kibana: <https://www.elastic.co/downloads/kibana>
- › **CURL without privileges:**
<http://www.confusedbycode.com/curl/#downloads>



INSTALLING and RUN ElasticSearch

› **Extract** ElasticSearch package (USING WinZip)

› Go inside the folder of elasticserach: **elasticsearch-7.16.1/bin**

› Fow Windows Users:

Hold **Shift + Right Click** in the folder and click
"open command window here"

› Execute in command: **elasticsearch.bat**

```
[2017-11-09T12:55:30,900][INFO ][o.e.n.Node               ] [] initializing ...
[2017-11-09T12:55:31,032][INFO ][o.e.e.NodeEnvironment ] [nHARRgJ] using [1] data paths, mounts [/ /dev/d
.1gb], net total_space [464.7gb], spins? [unknown], types [hfs]
[2017-11-09T12:55:31,033][INFO ][o.e.e.NodeEnvironment ] [nHARRgJ] heap size [1.9gb], compressed ordinary o
[2017-11-09T12:55:31,034][INFO ][o.e.n.Node               ] node name [nHARRgJ] derived from node ID [nHARRgJa
e] to override
[2017-11-09T12:55:31,035][INFO ][o.e.n.Node               ] version[5.6.4], pid[10059], build[8bbedf5/2017-10-
/10.12.6/x86_64], JVM[Oracle Corporation/Java HotSpot(TM) 64-Bit Server VM/1.8.0_101/25.101-b13]
[2017-11-09T12:55:31,035][INFO ][o.e.n.Node               ] JVM arguments [-Xms2g, -Xmx2g, -XX:+UseConcMarkSwe
cyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfi
=true, -Djdk.io.permissionsUseCanonicalPath=true, -Dio.netty.noUnsafe=true, -Dio.netty.noKeySetOptimization=tr
acityPerThread=0, -Dlog4j.shutdownHookEnabled=false, -Dlog4j2.disable.jmx=true, -Dlog4j.skipJansi=true, -XX:+H
s.path.home=/Users/kram/Desktop/MASTER/elasticsearch-5.6.4]
[2017-11-09T12:55:32,539][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [aggs-matrix-stats]
[2017-11-09T12:55:32,539][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [ingest-common]
[2017-11-09T12:55:32,539][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [lang-expression]
[2017-11-09T12:55:32,540][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [lang-groovy]
[2017-11-09T12:55:32,540][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [lang-mustache]
[2017-11-09T12:55:32,540][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [lang-painless]
[2017-11-09T12:55:32,542][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [parent-join]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [percolator]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [reindex]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [transport-netty3]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService     ] [nHARRgJ] loaded module [transport-netty4]
[2017-11-09T12:55:32,544][INFO ][o.e.p.PluginsService     ] [nHARRgJ] no plugins loaded
[2017-11-09T12:55:35,532][INFO ][o.e.d.DiscoveryModule   ] [nHARRgJ] using discovery type [zen]
[2017-11-09T12:55:36,638][INFO ][o.e.n.Node               ] initialized
[2017-11-09T12:55:36,638][INFO ][o.e.n.Node               ] [nHARRgJ] starting ...
[2017-11-09T12:55:37,075][INFO ][o.e.t.TransportService ] [nHARRgJ] publish_address {127.0.0.1:9300}, bound_
::1}:9300, {127.0.0.1:9300}
[2017-11-09T12:55:40,190][INFO ][o.e.c.s.ClusterService ] [nHARRgJ] new_master {nHARRgJ}{nHARRgJaRB6Bb4KCKTC
27.0.0.1}{127.0.0.1:9300}, reason: zen-disco-elected-as-master ([0] nodes joined)
[2017-11-09T12:55:40,262][INFO ][o.e.h.n.Netty4HttpServerTransport] [nHARRgJ] publish_address {127.0.0.1:9200}
200}, {[::1]:9200}, {127.0.0.1:9200}
[2017-11-09T12:55:40,262][INFO ][o.e.n.Node               ] [nHARRgJ] started
[2017-11-09T12:55:40,274][INFO ][o.e.g.GatewayService     ] [nHARRgJ] recovered [0] indices into cluster_state
[2017-11-09T13:15:09,966][INFO ][o.e.c.m.MetadataCreateIndexService] [nHARRgJ] [.kibana] creating index, cause
```

RUN Kibana

› **Extract** Kibana package (USING WinZip)

› Go inside the folder of kibana: **kibana-7.16.1-darwin-x86_64/bin**

› Fow Windows Users:

Hold **Shift + Right Click** in the folder and click
"open command window here"

› Execute in command: **kibana.bat**

```
[2017-11-09T12:55:30,900][INFO ][o.e.n.Node] [] initializing ...
[2017-11-09T12:55:31,032][INFO ][o.e.e.NodeEnvironment] [nHARRgJ] using [1] data paths, mounts [[/ (dev/disk1)]]], net usable_space [131
.1gb], net total_space [464.7gb], spins? [unknown], types [hfs]
[2017-11-09T12:55:31,033][INFO ][o.e.e.NodeEnvironment] [nHARRgJ] heap size [1.9gb], compressed ordinary object pointers [true]
[2017-11-09T12:55:31,034][INFO ][o.e.n.Node] [nHARRgJ] node name [nHARRgJ] derived from node ID [nHARRgJaRB6Bb4KCKtCmvg]; set [node.nam
e] to override
[2017-11-09T12:55:31,035][INFO ][o.e.n.Node] [nHARRgJ] version[5.6.4], pid[10059], build[8bbedf5/2017-10-31T18:55:38.105Z], OS[Mac OS X
/10.12.6/x86_64], JVM[Oracle Corporation/Java HotSpot(TM) 64-Bit Server VM/1.8.0_101/25.101-b13]
[2017-11-09T12:55:31,035][INFO ][o.e.n.Node] [nHARRgJ] JVM arguments [-Xms2g, -Xmx2g, -XX:+UseConcMarkSweepGC, -XX:CMSInitiatingOccupan
cyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djna.nosys
=true, -Djdk.io.permissionsUseCanonicalPath=true, -Dio.netty.noUnsafe=true, -Dio.netty.noKeySetOptimization=true, -Dio.netty.recycler.maxCap
acityPerThread=0, -Dlog4j.shutdownHookEnabled=false, -Dlog4j2.disable.jmx=true, -Dlog4j.skipJansi=true, -XX:+HeapDumpOnOutOfMemoryError, -De
s.path.home=/Users/kram/Desktop/MASTER/elasticsearch-5.6.4]
[2017-11-09T12:55:32,539][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [aggs-matrix-stats]
[2017-11-09T12:55:32,539][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [ingest-common]
[2017-11-09T12:55:32,539][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [lang-expression]
[2017-11-09T12:55:32,540][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [lang-groovy]
[2017-11-09T12:55:32,540][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [lang-mustache]
[2017-11-09T12:55:32,540][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [lang-painless]
[2017-11-09T12:55:32,542][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [parent-join]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [percolator]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [reindex]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [transport-netty3]
[2017-11-09T12:55:32,543][INFO ][o.e.p.PluginsService] [nHARRgJ] loaded module [transport-netty4]
[2017-11-09T12:55:32,544][INFO ][o.e.p.PluginsService] [nHARRgJ] no plugins loaded
[2017-11-09T12:55:35,532][INFO ][o.e.d.DiscoveryModule] [nHARRgJ] using discovery type [zen]
[2017-11-09T12:55:36,638][INFO ][o.e.n.Node] [nHARRgJ] initialized
[2017-11-09T12:55:36,638][INFO ][o.e.n.Node] [nHARRgJ] starting ...
[2017-11-09T12:55:37,075][INFO ][o.e.t.TransportService] [nHARRgJ] publish_address {127.0.0.1:9300}, bound_addresses {[fe80::1]:9300}, {[
::1]:9300}, {127.0.0.1:9300}
[2017-11-09T12:55:40,190][INFO ][o.e.c.s.ClusterService] [nHARRgJ] new_master {nHARRgJ}{nHARRgJaRB6Bb4KCKtCmvg}{nvV-sUUNRf-kD4GZjW0GfW}{1
27.0.0.1}{127.0.0.1:9300}, reason: zen-disco-elected-as-master ([0] nodes joined)
[2017-11-09T12:55:40,262][INFO ][o.e.h.n.Netty4HttpServerTransport] [nHARRgJ] publish_address {127.0.0.1:9200}, bound_addresses {[fe80::1]:9
200}, {[::1]:9200}, {127.0.0.1:9200}
[2017-11-09T12:55:40,262][INFO ][o.e.n.Node] [nHARRgJ] started
[2017-11-09T12:55:40,274][INFO ][o.e.g.GatewayService] [nHARRgJ] recovered [0] indices into cluster_state
[2017-11-09T13:15:09,966][INFO ][o.e.c.m.MetadataCreateIndexService] [nHARRgJ] [.kibana] creating index, cause [api], templates [], shards [
```

Excercise 1! Is It runnig?

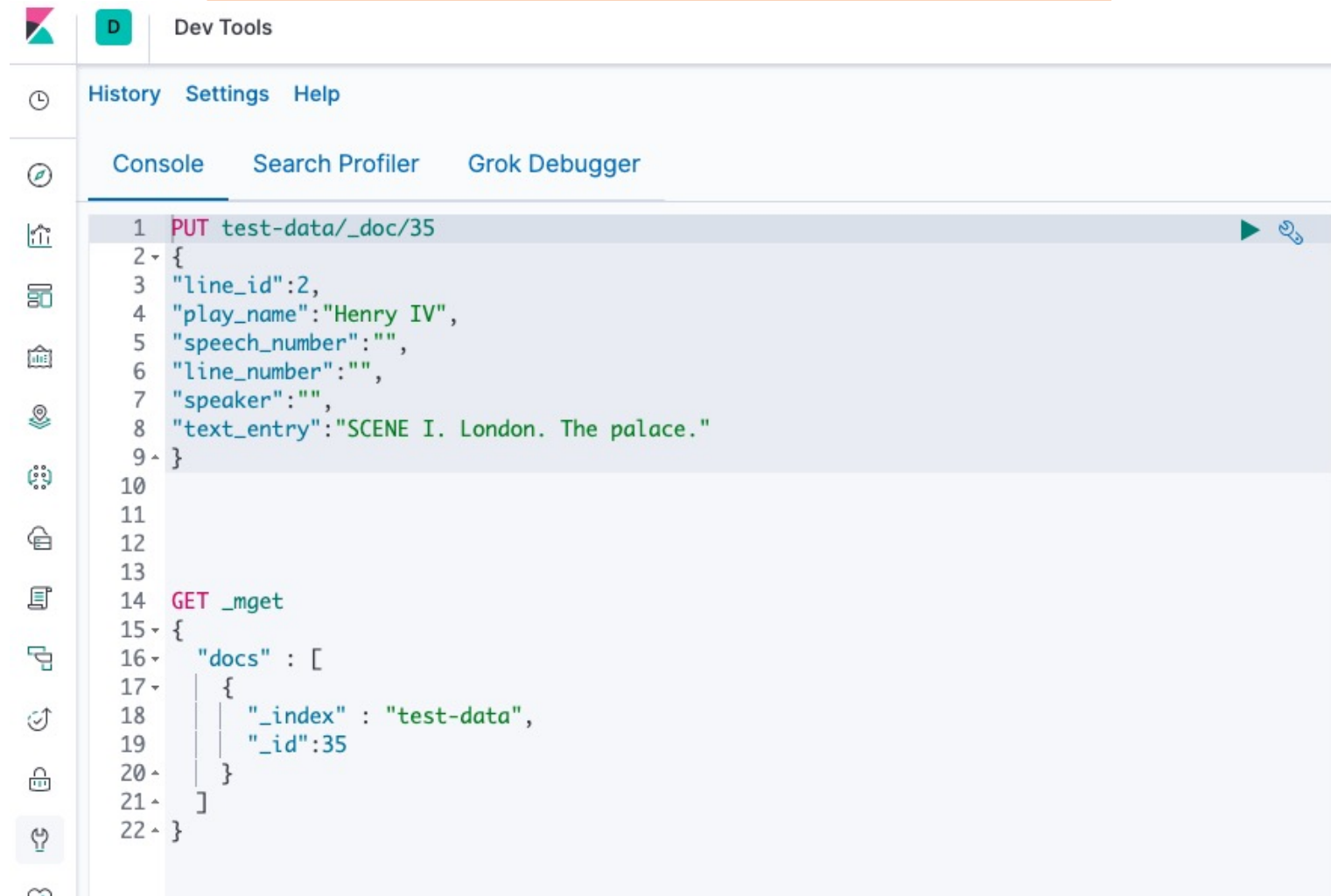
<http://localhost:9200/?pretty>

RESPONSE :

```
{
  "status" : 200,
  "name" : "elasticsearch",
  "version" : {
    "number" : "7.2.0",
    "build_hash" : "f1585f096d3f3985e73456debd1a0745f512bbc",
    "build_timestamp" : "2019-06-20T14:27:12Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0"
  },
  "tagline" : "You Know, for Search"
}
```


Excercise 2! Is Kibana runnig?

Visit: <http://localhost:5601>



The screenshot shows the Kibana DevTools interface. The 'Console' tab is active, displaying two REST API requests and their responses. The first request is a PUT to test-data/_doc/35 with a JSON body. The second request is a GET _mget which returns a list of documents, including the one just created.

```
1 PUT test-data/_doc/35
2 {
3   "line_id":2,
4   "play_name":"Henry IV",
5   "speech_number":"",
6   "line_number":"",
7   "speaker":"",
8   "text_entry":"SCENE I. London. The palace."
9 }
10
11
12
13
14 GET _mget
15 {
16   "docs" : [
17     {
18       "_index" : "test-data",
19       "_id":35
20     }
21   ]
22 }
```

Try to add a document

In KIBANA open the developer TAB, copy the following code and run it!

What is the answer received by the Server?

```
PUT test-data/_doc/21
{
  "rank": 21,
  "city": "Boston",
  "state": "Massachusetts",
  "population2010": 617594,
  "land_area": 48.277,
  "location": {"lat": 42.332, "lon": 71.0202 },
  "abbreviation": "MA"
}
```

Retrieve the DOCUMENT!

Request in Kibana Developers Tools:

GET test-data/_doc/21

RESPONSE:

```
{ "_index": "test-data", "_type": "cities", "_id": "21", "_version": 8, "found": true,
  "_source": { "rank": 21, "city": "Boston", "state": "Massachusetts",
    "population2010": 617594, "land_area": 48.277, "location": {"lat": 42.332,
      "lon": 71.0202}, "abbreviation": "MA" }}
```

Exercise 3 Add a new DOCUMENT!

Add the following document into the index: [test-data](#) and ID: [35](#)

Request in Kibana Developers Tools:

```
PUT ?/_doc/?  
{  
  "line_id":2,  
  "play_name":"Henry IV",  
  "speech_number": "",  
  "line_number": "",  
  "speaker": "",  
  "text_entry":"SCENE I. London. The palace."  
}
```

DELETE a document!

Request in Kibana Developers Tools:

DELETE test-data/_doc/21

OUTPUT:

```
{  
  "found": true,  
  "_index": "test-data",  
  "_type": "cities",  
  "_id": "21",  
  "_version": 13,  
  "result": "deleted",  
  "_shards": { "total": 2, "successful": 1, "failed": 0 }  
}
```

What about the index???

LOAD DATA as BATCH



Loading Sample Data

The complete works of William Shakespeare, suitably parsed into fields.

Download this data set by clicking here: [shakespeare.json](#)

A set of fictitious accounts with randomly generated data.

Download this data set by clicking here: [accounts.zip](#)

A set of randomly generated log files.

Download this data set by clicking here: [logs.jsonl.gz](#)

How are the files Formatted?

LOAD DATA as BATCH



Shakespeare.json

```
{  
  "line_id": INT,  
  "play_name": "Keyword",  
  "speech_number": INT,  
  "line_number": "Keyword",  
  "speaker": "Keyword",  
  "text_entry": "Text"  
}
```

Accounts.json

```
{  
  "account_number": INT,  
  "balance": INT,  
  "firstname": "Keyword",  
  "lastname": "Keyword",  
  "age": INT,  
  "gender": "M or F",  
  "address": "Text",  
  "employer": "Keyword",  
  "email": "Keyword",  
  "city": "Keyword",  
  "state": "Keyword"  
}
```

Logs.jsonl

```
{  
  "memory": INT,  
  "geo.coordinates": "geo_point",  
  "@timestamp": "date"  
}
```

Create Explicitly the Indexes!

PUT /shakespeare

```
{
  "mappings" : {
    "properties" : {
      "speaker" : {"type": "keyword" },
      "play_name" : {"type": "keyword" },
      "line_id" : { "type" : "integer" },
      "speech_number" : { "type" : "integer" },
      "text_entry" : {"type": "text" }
    }
  }
}
```

PUT /logstash-2015.05.18

```
{
  "mappings": {
    "properties": {
      "geo": {
        "properties": {
          "coordinates": { "type": "geo_point" }
        }
      },
      "ip" : {"type": "ip" },
      "@timestamp" : {"type": "date" }
    }
  }
}
```

PUT /logstash-2015.05.19

PUT /logstash-2015.05.20

PUT /bank

```
{
  ???????
}
```

CURL LOADER

- 1) Go into the folder containing the datasets
- 2) Open the *command terminal*
- 3) User **CURL** commands for storing DATA
- 4) Verify successful loading with the following command:
GET */_cat/indices?v*

The Elasticsearch bulk API loads the data sets with the following commands:

```
curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/bank/_doc/_bulk?pretty' --data-binary @accounts.json
```

```
curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/shakespeare/_bulk?pretty' --data-binary @shakespeare.json
```

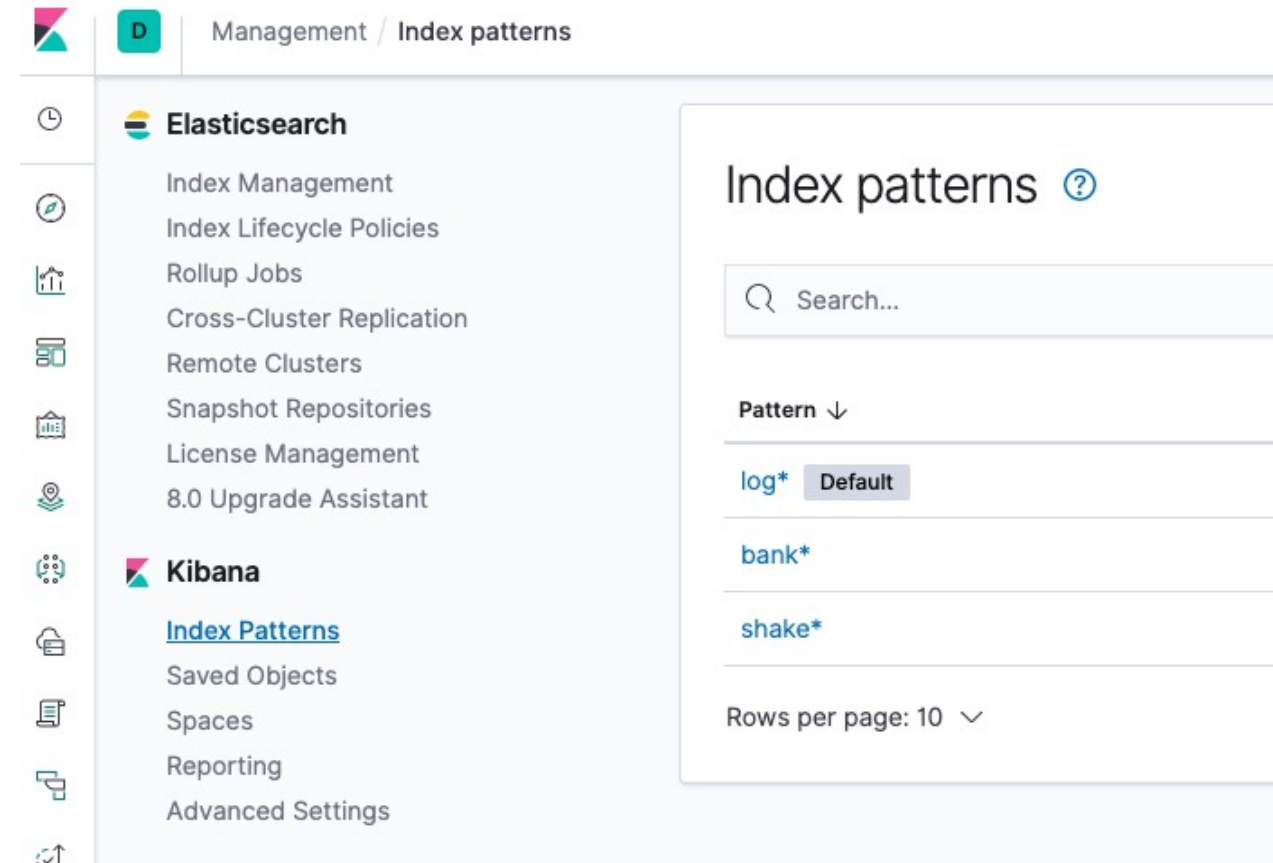
```
curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/_bulk?pretty' --data-binary @logs.jsonl
```

Define KIBANA Index Patterns

Specify **shakes*** as the index pattern for the Shakespeare dataset.

Then define a second index pattern named **bank***

The Logstash data set does contain *time-series data*, so after clicking Add New to define the index pattern **log*** for this data set, make sure the **Time Filter field** is setted to **@timestamp**





SEARCHER

Visualize data in Kibana

You can construct **searches by using the field** names and the values you're interested in.

With **numeric fields** you can use comparison operators such as greater than **>**, less than **<**, or equals **=**. You can link elements with the logical operators **AND**, **OR**, and **NOT**, all in uppercase.

To try it out, select the **bank* index pattern** and enter the following query string in the query bar:

```
account_number:<100 AND balance:>47500
```

Results

 **D** Discover 

5 hits

New Save Open Share Inspect

Filters

account_number:<100 AND balance:>47500

Lucene

Refresh

bank*

Selected fields

? _source

Available fields

t _id

t _index

_score

t _type

account_number

t address

age

balance

t city

Source

>

account_number: 32 balance: 48,086 firstname: Dillard lastname: Mcpherson age: 34 gender: F address: 702 Quentin Street employer: Quailcom email: dillardmcpherson@quailcom.com city: Veguita state: IN _id: 32 _type: account _index: bank _score: 2

>

>

account_number: 78 balance: 48,656 firstname: Elvira lastname: Patterson age: 23 gender: F address: 834 Amber Street employer: Assistix email: elvirapatterson@assistix.com city: Dunbar state: TN _id: 78 _type: account _index: bank _score: 2

>

>

account_number: 85 balance: 48,735 firstname: Wilcox lastname: Sellers age: 20 gender: M address: 212 Irving Avenue employer: Confrenzy email: wilcox sellers@confrenzy.com city: Kipp state: MT _id: 85 _type: account _index: bank _score: 2

>

>

account_number: 97 balance: 49,671 firstname: Karen lastname: Trujillo age: 40 gender: F address: 512 Cumberland Walk employer: Tsunamia email: karentrujillo@tsunamia.com city: Fredericktown state: MO _id: 97 _type: account _index: bank _score: 2

>

>

account_number: 8 balance: 48,868 firstname: Jan lastname: Burns age: 35 gender: M address: 699 Visitation Place employer: Glasstep email: janburns@glasstep.com city: Wakulla state: AZ _id: 8 _type: account _index: bank _score: 2

Query Syntax

A query clause used in query context answers the question

“How well does this document match this query clause?”

Besides deciding whether or not the document matches, the query clause also calculates a **_score** representing how well the query matches other documents

The most simple query, which matches all documents, giving them all a **_score** of 1.0.

```
GET bank/_search
{
  "query": {
    "match_all": {}
  }
}
```

Match Queries!

The high-level **full text queries** are usually used for running queries on **full text fields** like the body of an email.

The **match** query is of type **boolean**. It means that the text provided is analyzed and the analysis process constructs **a boolean query** from the provided text.

```
GET /_search
{
  "query": {
    "match": {
      "text_entry": {
        "query": "knife war edge",
        "operator": "and"
      }
    }
  }
}
```

Phrase Queries!

A **phrase query** will match documents that contain the given list of terms in **any order**, and at a **maximum edit distance of slop**

```
GET /_search
{
  "query": {
    "match_phrase" : {
      "text_entry" : {
        "query" : "this is a sight",
        "slop" : 10
      }
    }
  }
}
```

I don't remember the...

Match Phrase Prefix Query

The `match_phrase_prefix` is the same as `match_phrase`, except that it allows for prefix matches on the last term in the text. For example:

```
GET /_search
{
  "query": {
    "match_phrase_prefix": {
      "text_entry": {
        "query": "this is p",
        "slop": 10,
        "max_expansions": 3
      }
    }
  }
}
```

Muti-fields

The `multi_match` query builds on the `match query` to allow multi-field queries:

```
GET /_search
{
  "query": {
    "multi_match" : {
      "query" : "Street",
      "type": "best_fields",
      "fields": ["text_entry", "address^999"]
    }
  }
}
```

best_fields	(default) Finds documents which match any field, but uses the <code>_score</code> from the best field.
most_fields	Finds documents which match any field and combines the <code>_score</code> from each field.
cross_fields	Treats fields with the same analyzer as though they were one big field. Looks for each word in any field.
phrase	Runs a <code>match_phrase</code> query on each field and combines the <code>_score</code> from each field.

Stopwords and CUT-off

The match query supports a **cutoff_frequency** that allows specifying an absolute or relative **document frequency** where **high frequency terms are moved into an optional subquery**.

Terms used in the **score only if** they are **lower the cut-off frequency** (below the cutoff)

```
GET /_search
{
  "query": {
    "match": {
      "text_entry": {
        "query": "this is the man",
        "cutoff_frequency": 0.001,
        "operator": "or"
      }
    }
  }
}
```


Full String Queries

Differently than before, the request is written directly in the query string

The research can be executed in specific fields or in all the indexes, fields and documents.

```
GET /_search
{
  "query": {
    "query_string" : {
      "query" : "this AND is OR man"
    }
  }
}
```

```
GET /_search
{
  "query": {
    "query_string" : {
      "fields":["text_entry^3","address"],
      "query" : "this AND is OR man"
    }
  },
  "_source": "text_entry"
}
```

Full String Queries Operators

- **Field names**

(text_entry: this OR address:thus) AND (address:that OR account_number:25)

- Wildcards

qu?ck bro* ; speaker:*

- Regular Expression

name:/joh?n(ath[oa]n)/

- Fuzziness

uikc~ brwn~ foks~ Damerau-Levenshtein distance

- Ranges

date:[2012-01-01 TO 2012-12-31] AND age:(>=10 AND <20)

- Boost

"john smith"^2

- **Boolean Operators**

quick brown +fox -news

Term Queries

You can use the term query to find documents based on a **precise value** such as a price, a product ID, or a username.

```
GET /_search
{
  "query":
    { "term":
      { "text_entry":
        { "value": "man",
          "boost": 1.0
        }
      }
    }
}
```

Exercise?



1. Find all the Shakespeare phrases for which the speaker is a “KING” but not “HENRY IV”
2. Find the Shakespeare phrase which contain “to be or not to be”
3. Show all the logs in the interval **May 1st 2015, 00:00:00.000 - May 31st 2015, 00:00:00.000** and agent something like “Mozilla”
4. Show the Name and Surname of the bank accounts with a balance higher than 40000, age less than 30 and name not Johnson
5.

A
Ω
·
T·
·
5 6 7 4 3 2 1
ELIAKIM.

Filio Helk I HV, Esai. 22.v.20.
Angelo Phila Delphiae ApO. 3. v. 7.

(In this ARabian desert, not far from the city of Saba of the MAgi, where CHRIST by an act of the Spirit, after his baptism, fasted for 40 days & 40 nights, &, tried THREE TIMES by the Devil, and TRIUMPED).
Matth.4 Mark.I Luke.4. ApOcA. 3.v,10.

**IN SPIRIT!
IN TRUTH!**

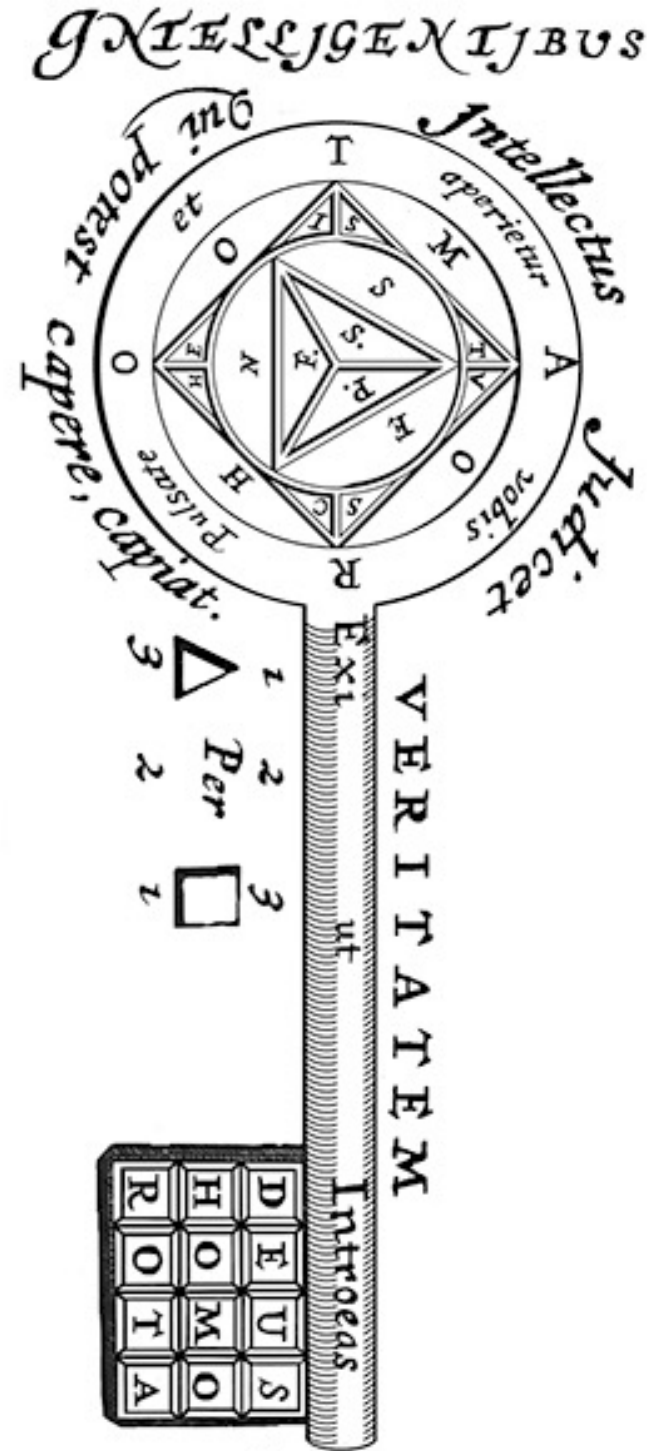
Who has the K E Y of David:
Who opens, & no one will close:
Who closes, & no one will open.
Job 12,14. Esa.22,22.

S
A + O
M D C
X X X
V V V

D. 25. M. Mart.I.

In VR (⌚) cujus

Insignia, (⌚) Coronata.



Luca x1.52
*VAE Vobis Legis peritis, quia tulistis CLA:
VEM SCIENTIAE: ipsi non introistis, et
introeuntes prohibuistis.*