# ElasticSearch

* Elasticsearch is a real-time distributed search and analytics engine.
* It is used for full-text search, structured search, analytics, and all three in combination
* Who are using - Wikipedia, stackoverflow, GitHub
* Elasticsearch is an open-source search engine built on top of Apache Lucene
* **What is ApacheLucene..?**  - Apache LuceneTM is a high-performance, full-featured text search engine library written entirely in Java.

It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

* Elasticsearch is written in Java and uses Lucene internally for all of its indexing and searching, but it aims to make full-text search easy by hiding the complexities of Lucene behind a simple, coherent, RESTful API.
* A distributed real-time document store where every field is indexed and searchable
* A distributed search engine with real-time analytics
* Capable of scaling to hundreds of servers and petabytes of structured and unstructured data
* **Cluster** is a group of nodes
* **Node** is a running instance of Elasticsearch
* A single node, however, can form a cluster all by itself.
* Elasticsearch is document oriented, meaning that it stores entire objects or documents. It not only stores them, but also indexes the contents of each document in order to make them searchable. In Elasticsearch, you index, search, sort, and filter documents—not rows of columnar data. This is a fundamentally different way of thinking about data and is one of the reasons Elasticsearch can perform complex full-text search.
* Elasticsearch uses JSON(JavaScript Object Notation) as the serialization format for documents.
* The act of storing data in Elasticsearch is called indexing, but before we can index a document, we need to decide where to store it
* An Elasticsearch cluster can contain multiple indices, which in turn contain multiple types. These types hold multiple documents, and each document has multiple fields.

- Practice –

* Index a document per employee, which contains all the details of a single employee.
* Each document will be of type employee.
* That type will live in the megacorp index.
* That index will reside within our Elasticsearch cluster.
* Crerating an index and document -

PUT /megacorp /employee/1

{

"first\_name" : "Abhay",

"last\_name" : "Pandey",

"age" : 25,

"about" : "I love to talkin on phone",

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

}

* megacorp - The index name
* employee - The type name
* 1 - The ID of this particular employee
* Retrieving a document -
* GET /megacorp/employee/1
* Deleting a document -
* Delete /megacorp/employee/1
* Documnet is exist -
* HEAD /megacorp/employee/1
* GET /megacorp/employee/\_search
* Searches all employees of type 'employee'
* GET /megacorp/\_search
* Searches all including all types
* GET /megacorp/employee/\_search?q=last\_name:Pandey
* **- Query DSL** - Elasticsearch provides a rich, flexible, query language called the query DSL, which allows us to build much more complicated, robust queries. The domain-specific language (DSL) is specified using a JSON request body.
  + - ex - GET /megacorp/employee/\_search

{

"query" : {

"match" : {

"last\_name" : "Smith"

}

}

}

* + - ex - GET /megacorp/employee/\_search

{

"query" : {

"bool" : {

"must" : {

"match" : {

"last\_name" : "smith"

}

},

"filter" : {

"range" : {

"age" : { "gt" : 30 }

}

}

}

}

}

* + - ex - GET /megacorp/employee/\_search

{

"query" : {

"match" : {

"about" : "rock climbing"

}

}

}

* By default, Elasticsearch sorts matching results by their relevance score, that is, by how well each document matches the query. The first and highest-scoring result comes first
* Score would we less if exact word is not in documnet, but result will contain this
* Elasticsearch can search within full-text fields and return the most relevant results first. This concept of relevance is important to Elasticsearch, and is a concept that is completely foreign to traditional relational databases, in which a record either matches or it doesn’t.
* Phrase Search -
  + - Finding individual words in a field is all well and good, but sometimes you want to match exact sequences of words or phrases. For instance, we could perform a query that will match only employee records that contain both “rock” and “climbing” and that display the words next to each other in the phrase “rock climbing.”
* GET /megacorp/employee/\_search
  + - * {
      * "query" : {
        + "match\_phrase" : {
        + "about" : "rock climbing"
        + }
      * }
      * }
* Elasticsearch has functionality called aggregations, which allow you to generate sophisticated analytics over your data. It is similar to GROUP BY in SQL, but much more powerful.
* ex - GET /megacorp/employee/\_search
  + - * {
      * "aggs": {
      * "all\_interests": {
      * "terms": { "field": "interests" }
      * }
      * }
      * }
* ex - GET /megacorp/employee/\_search
  + - * {
      * "query": {
      * "match": {
      * "last\_name": "smith"
      * }
      * },
      * "aggs": {
      * "all\_interests": {
      * "terms": {
        + "field": "interests"
      * }
      * }
      * }
      * }
* Aggregations allow hierarchical rollups too.
* ex - GET /megacorp/employee/\_search
  + - * + {
        + "aggs" : {

"all\_interests" : {

"terms" : { "field" : "interests" },

"aggs" : {

"avg\_age" : {

"avg" : { "field" : "age" }

}

}

}

* + - * + }
        + }