Search Objective:

Find the most relevant documents with your search terms.

Steps:

1. Know of the document’s existence.
2. Index the document for lookup.
3. Know how relevant the documents is.
4. Retrieve ranked by relevance.
5. Web crawler is used to gather all the documents and links over the net and create a corpus of documents.
6. All pages searched by web crawler are tokenized. Every word and its corresponding document is stored in inverted index data structure. Inverted document is the mapping between the term and the document where the term is found. It is inverted because it goes from the search term to the web page document.
7. Scoring is associated with the document where term found is more as compared to another document. More the scoring that document will be on top of the search.
8. Search it will refer to the inverted index to find the relevant document for the specific term.

Inverted Index:

* Tokenize the text into words.
* Prepare a table for the frequency of terms present in the corpus of documents.

Apache Lucene:

The indexing and search library for a high performance, fill-text search engine.

Like Hadoop in a distributed computing world, Lucene acts as nucleus of several technologies built around it.

Technologies built on top of Apache Lucene are:

* Solr: A search server with distributed indexing, load balancing, replication, automated recover, centralized configuration.
* Nutch: Web crawler and index parsing which can be used along with Apache Lucene to make a complete search engine.
* CrateDB: Open Source, SQL distributed database.

Elastic Search:

It is distributed search and analytic engine built on Apache Lucene.

Characteristics:

* Distributed: Scales to thousands of nodes.
* High Availability: Multiple copies of data.
* Restful API: CRUD, monitoring and other operation via simple JSON based HTTP calls.
* Powerful Query DSL: Express complex queries simply.
* Schemaless: Index data without an explicit schema.

Use Cases:

Distributed Search

* Product Catalog, Inventory, Autocomplete.
* Video Clips, Categories, Tags.
* Courses, Authors, Topics.

Analytics Engine

* Mining Log Data for Insights.
* Price alerting platforms.
* Business analytics and Intelligence.

Checks on Elastic Search:

* Curl -XGET ‘localhost:9200/\_cat/indices?v&pretty’
* Curl -XGET ‘localhost:9200/\_cat/nodes?v&pretty’
* Curl -XGET ‘localhost:9200/\_cat/health?v&pretty’

Create Indices:

* Curl -XPUT ‘localhost:9200/products?&pretty’

Curl -XPUT ‘localhost:9200/customers?&pretty’

Curl -XPUT ‘localhost:9200/orders?&pretty’

This will create index name as products. XPUT is used to create new resources.

Pri column represents shards.

Rep column represents replication.

Health is yellow, which means it has only one node.

Docs.count represent number of documents.

5 shards 1 replica is default setting for an index in Elastic Search.

Adding documents to an index:

* Curl -XPUT ‘localhost:9200/products/mobiles/1?pretty’ -d’

{

“name”: “I phone 7”,

“camera”: “12MP”,

“storage”: “256GB”,

“display”: “4.7inch”,

“battery”: “1,960mAH”,

“reviews”: [“Incredibly happy after having used it for one week”, “Best iphone so far”, “Very expensive, stick to Android”]

}

‘

Retrieve documents from elastic search by id:

* Full document:

Curl -XGET ‘localhost:9200/products/mobiles/1?pretty’

* Partial document:

Curl -XGET ‘localhost:9200/products/mobiles/1?pretty&\_source=False’

Curl -XGET ‘localhost:9200/products/mobiles/1?pretty&\_source=name,reviews’

Update documents by id:

* Full document:

Curl -XPUT ‘localhost:9200/products/mobiles/3?pretty’ -d’

{

“name”: “Xiaomi Note”,

“camera”: “12MP”,

“storage”: “256GB”,

“display”: “5.5inch”,

“battery”: “1,800mAH”,

“reviews”: [“Really Love Xiaomi Products”, “Too large to use easily”]

}

‘

* Partial Document Updates:

\_update API is used, and also use the POST command with a “doc” field.

Curl -XPOST ‘localhost:9200/products/mobiles/2/\_update?pretty’ -d’

{

“doc”: {

“color”: “black”

}

}

‘

Curl -XPOST ‘localhost:9200/products/mobiles/2/\_update?pretty’ -d’

{

“doc”: {

“reviews”: [“Samsung is my Favorite”],

“texture”: “smooth”

}

}

‘

Using script tag to update document: Make sure to add below line

script.engine.groovy.inline.update: on in the file elasticsearch.yml to make it work.

Curl -XPOST ‘localhost:9200/products/shoes/1/\_update?pretty’ -d’

{

“script”: “ctx.\_source.size += 2”

}

’

Deleting Documents and Indices:

Curl -XDELETE ‘localhost:9200/products/mobiles/2?pretty’

* To check whther document exist or not use below command:

Curl -I -XHEAD ‘localhost:9200/products/mobiles/2?pretty’ will result in 404 Not found error.

* Delete index:

Curl -XDELETE ‘localhost:9200/customers?pretty’

Fetch Multiple Documents from single query:

Curl -XGET ‘localhost:9200/\_mget?pretty’ -d’

{

“docs”:

[{

“\_index”: “products”,

“\_type”: “laptops”,

“\_id”: “1”

},

{

“\_index”: “products”,

“\_type”: “laptops”,

“\_id”: “2”

}]

}’

OR

Curl -XGET ‘localhost:9200/products/\_mget?pretty’ -d’

{

“docs”:

[{

“\_type”: “laptops”,

“\_id”: “1”

},

{

“\_type”: “laptops”,

“\_id”: “2”

}]

}’

OR

Curl -XGET ‘localhost:9200/products/laptops/\_mget?pretty’ -d’

{

“docs”:

[{

“\_id”: “1”

},

{

“\_id”: “2”

}]

}’

BULK Operations:

Curl -XPOST ‘localhost:9200/\_bulk?pretty’ -d’

{ “index” : { “\_index” : “products”, “\_type” : “shoes”, “\_id” : “3” } }

{ “name” : “Puma”, “size”: 9, “color” : “black” }

{ “index” : { “\_index” : “products”, “\_type” : “shoes”, “\_id” : “4” } }

{ “name” : “New Balance”, “size”: 8, “color” : “black” }

Below will generate id automatically:

Curl -XPOST ‘localhost:9200/products/shoes/\_bulk?pretty’ -d’

{ “index” : { } }

{ “name” : “Puma”, “size”: 9, “color” : “black” }

{ “index” : { } }

{ “name” : “New Balance”, “size”: 8, “color” : “black” }

Another example:

Curl -XPOST ‘localhost:9200/products/shoes/\_bulk?pretty’ -H ‘Content-Type: application/json’ -d’

{ “index” : { “\_id” : “3” } }

{ “name” : “Puma”, “size”: 9, “color” : “black” }

{ “index” : { “\_id” : “4” } }

{ “name” : “New Balance”, “size”: 8, “color” : “black” }

{ “delete” : { “\_id” : “2” } }

{ “create” : {“\_id” : “5” } }

{ “name” : “Nike Power”, “size” : 9, “color” : “black” }

{ “update” : { “\_id” : “1” } }

{ “doc” : { “color” : “orange” } }

‘

Bulk Indexing of Documents from JSON file:

Json File:

{ “index” : { } }

{ “name” : “Carlson Barnes”, “age”: 34 }

{ “index” : { } }

{ “name” : “Sheppard Stein”, “age”: 39 }

{ “index” : { } }

{ “name” : “Nixon Singleton”, “age”: 36 }

{ “index” : { } }

{ “name” : “Sharron Sosa”, “age”: 33 }

{ “index” : { } }

{ “name” : “Kendra Cabrera”, “age”: 24 }

{ “index” : { } }

{ “name” : “Young Robinson”, “age”: 20 }

Query Used:

Curl -H “Content-Type: application/x-ndjson” -XPOST ‘localhost:9200/customers/personal/\_bulk?pretty&refresh’ –data-binary @”customers.json”

Search in Elastic search have Two Contexts of Search:

* How well does this document match this query? Query Context
* Does this document match this query clasue? Filter Context

Query Context:

* Included or not: Determine whether the document should be part of the result.
* Relevance score: Calculated for every search term the document maps to.
* High score, more relevant: More relevant documents, higher in the search rankings.

Filter Context:

* Included or not: Yes/no determines whether included in the result.
* No scoring: No additional relevance ranking in the search results.
* Structured data: Exact matches, range queries.
* Faster: Only determine inclusion in results, no scoring to consider.

JSON-Generator:

[

‘{{repeat(1000, 1000)}}’,

{

Name: ‘{{firstName()}} {{surname()}}’,

Age: ‘{{integer(18, 75)}}’,

Gender: ‘{{gender()}}’,

Email: ‘{{email()}}’,

Phone: ‘+1 {{phone()}}’,

Street: ‘{{integer(100, 9999)}} {{street()}}’,

City: ‘{{city()}}’,

State: ‘{{state()}}, {{integer(100, 10000)}}’

}

]

Delete the customer indices.

Command to load data from JSON customer file:

Curl -H “Content-Type: application/x-ndjson” -XPOST ‘localhost:9200/customers/personal/\_bulk?pretty&refresh’ –data-binary @”customers\_full.json”

Query Context Approach: Two ways of searching terms under Query Context:

* Search Terms as URL query parameters.

Curl -XGET ‘localhost:9200/customers/\_search?q=Wyoming&pretty’ : Whole document with Wyoming text. This is act as a http request.

Curl -XGET ‘localhost:9200/customers/\_search?q=Wyoming&sort=age:desc&pretty’

Curl -XGET ‘localhost:9200/customers/\_search?q=state:kentucky&from=10&size=2&pretty’

Curl -XGET ‘localhost:9200/customers/\_search?q=state:kentucky&explain&pretty’ : Explain means why the relevance score for this search has the value it did.

* Search Terms within the URL request body.

Curl -XGET ‘localhost:9200/products/\_search?pretty’ -d’

{

“query”: { “match\_all”: {} }

}

‘

Curl -XGET ‘localhost:9200/products/\_search?pretty’ -d’

{

“query”: { “match\_all”: {} },

“size”: 3

}

‘

Curl -XGET ‘localhost:9200/products/\_search?pretty’ -d’

{

“query”: { “match\_all”: {} },

“from”: 5,

“size”: 3

}

‘

To search multiple indices:

Curl -XGET ‘localhost:9200/customers,products/\_search?pretty’

To search multiple indices:

Curl -XGET ‘localhost:9200/products/shoes,laptops/\_search?pretty’

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“query”: { “match\_all”: {} },

“sort”: { “age” : { “order” : “desc” } },

“size” : 20

}

‘

Source Filtering to include only those fields that we are interested in:

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“query” : {

“term” : { “name” : “gates” }

}

}

‘

Term should have an exact match in the inverted index.

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“source” : false,

“query” : {

“term” : { “street” : “chestnut” }

}

}

‘

Will not show source values.

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“source” : “st\*”,

“query” : {

“term” : { “street” : “chestnut” }

}

}

‘

Will show source value starting with “st\*”. Source filtering does not affect relevance ranking in anyway.

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“source” : {

“includes” : [“st\*”, “\*n\*” ],

“excludes” : [ “\*der” ]

},

“query” : {

“term” : { “state” : “Washington” }

}

}’

FULL Text Queries Using:

Match, match\_phrase, match\_phrase\_prefix.

MATCH: use not for an exact match.

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“query” : {

“match” : {

“name” : “webster”

}

}

}

‘

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“query” : {

“match” : {

“name” : {

“query” : “frank norris”,

“operator” : “or”

}

}

}

}’

All documents with “frank” or “Norris” in the “name” field. If any operator is not mentioned, then by default the operator is “or”.

MATCH\_PHRASE: Will search for full phrase provided under street column having value as “Tompkins Place”

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“query” : {

“match\_phrase” : {

“street” : “Tompkins place”

}

}

}’

MATCH\_PHRASE\_PREFIX: Will search for name column starting with “ma”.

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“query” : {

“match\_phrase\_prefix” : {

“name” : “ma”

}

}

}’

Curl -XGET ‘localhost:9200/customers/\_search?pretty’ -d’

{

“query” : {

“match\_phrase\_prefix” : {

“street” : “Clymer st”

}

}

}’