Elastic Search

For BEGINERS

What is Elastic Search?

Elastic Search is a free and open source distributed inverted index created by Shay Banon.

Build on top of Apache Lucene

- Lucene is a most popular java-based full text search index implementation. First public release was in February 2010.

Developed in Java, so inherently cross-platform.

Why Elastic Search?

Easy to scale

Everything is one JSON call away (RESTful API)

Unleashed power of Lucene under the hood

Excellent Query DSL

Support for advanced search features (Full Text)

Configurable and Extensible

Document Oriented

Schema free

Conflict management

Active community

Easy to Scale

Elasticsearch allows you to start small, but will grow with your business. It is built to scale horizontally out of the box. As you need more capacity, just add more nodes, and let the cluster reorganize itself to take advantage of the extra hardware.
Elasticsearch achieves horizontal scalability by sharding its index and assigning each shard to a node in the cluster.

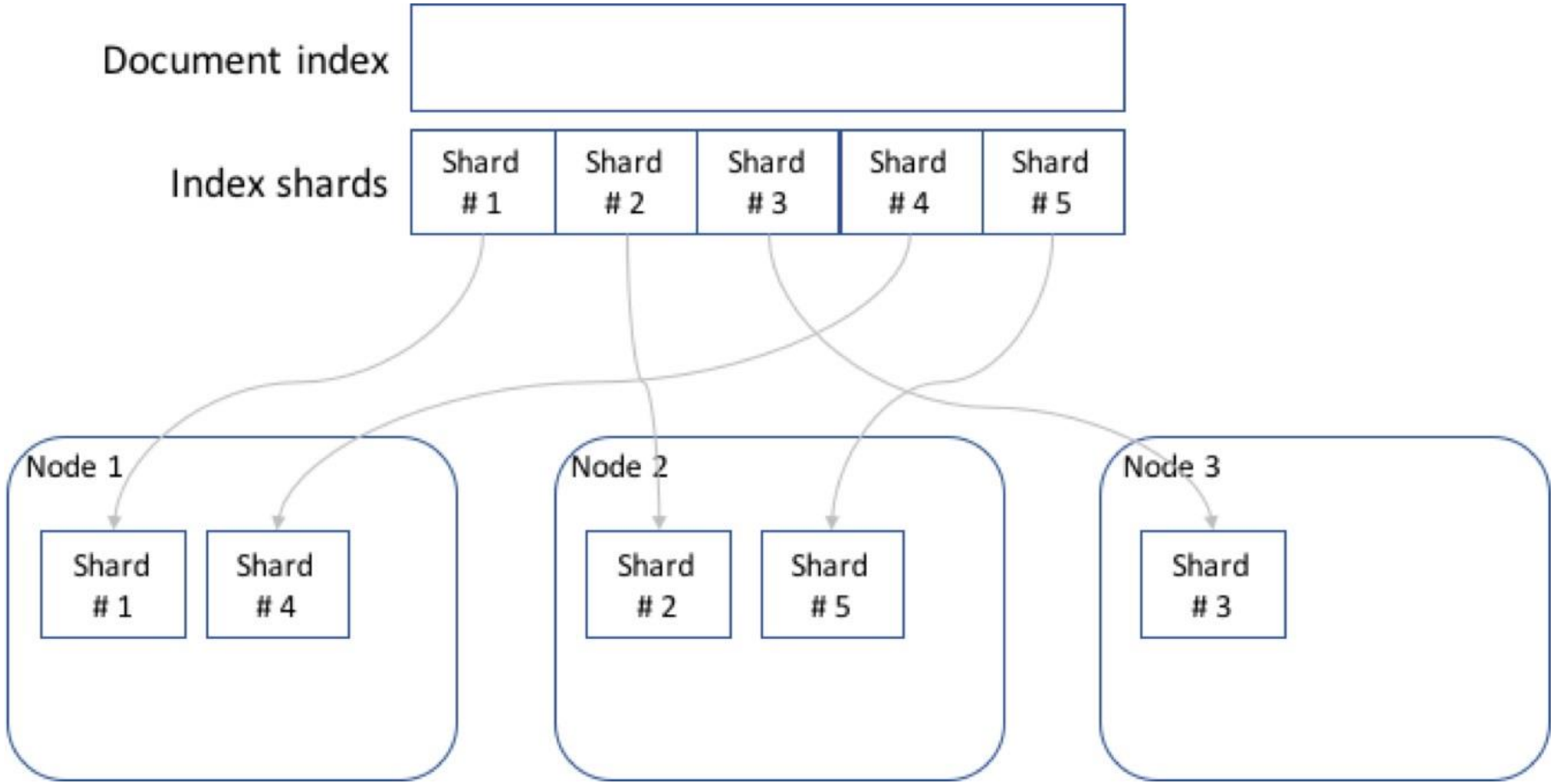


Figure 1: Sharding of the document index and assignment to nodes: 5 shards

Scaling horizontally allows each node to have to deal with only part of the full document index.
Furthermore, Elasticsearch also has the concept of *replicas*, which are copies of shards.
Figure 2 shows a cluster that can handle the loss of any one node as another node will have a replica of any of its primary shards.

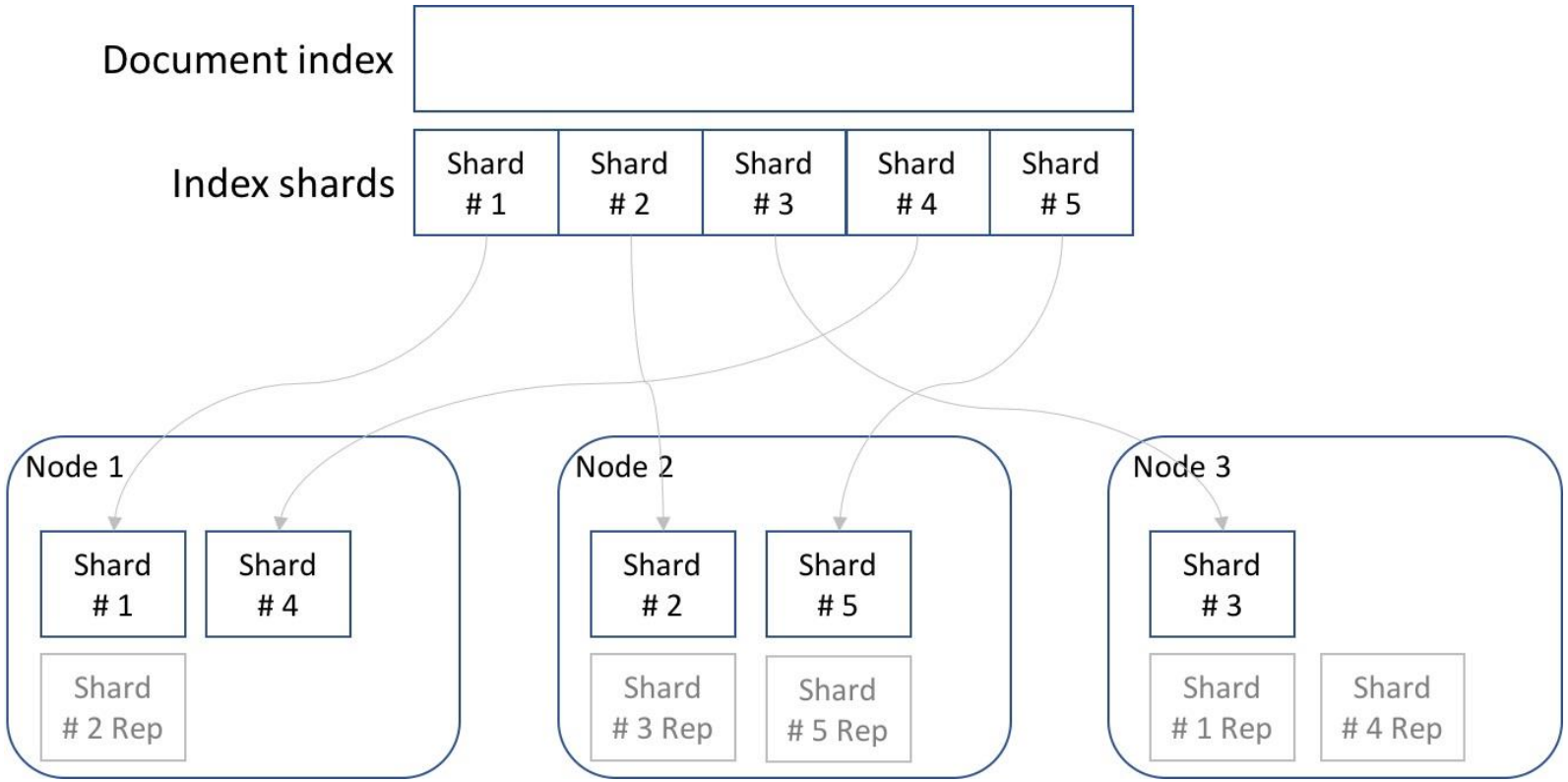
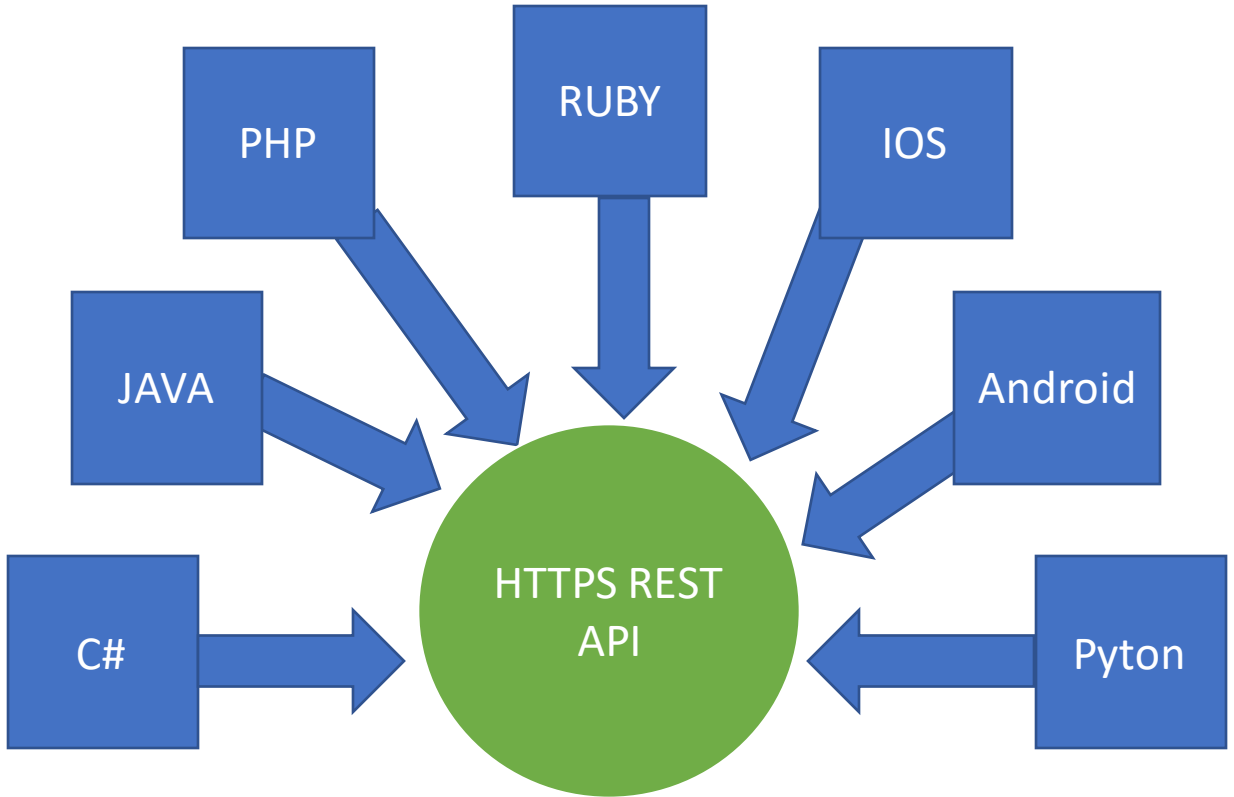


Figure 2: Shards and replicas — 5 shards, 1 replica

RESTful API

Elasticsearch is API driven. Almost any action can be performed using a simple RESTful API using JSON over HTTP. An API already exists in the language of your choice. Responses are always in JSON, which is both machine and human readable.



Per-operation Persistence

Elasticsearch puts your data safety first. Document changes are recorded in transaction logs on multiple nodes in the cluster to minimize the chance of any data loss.

Excellent Query DSL

he REST API exposes a very complex and capable query DSL, that is very easy to use. Every query is just a JSON object that can practically contain any type of query, or even several of them combined.

Using filtered queries, with some queries expressed as Lucene filters, helps leverage caching and thus speed up common queries, or complex queries with parts that can be reused.

Support for advanced search features (Full Text)

Elasticsearch uses Lucene under the covers to provide the most powerful full text search capabilities available in any open source product.

Search comes with multi-language support, a powerful query language, support for geolocation, context aware did-you-mean suggestions, autocomplete and search snippets.

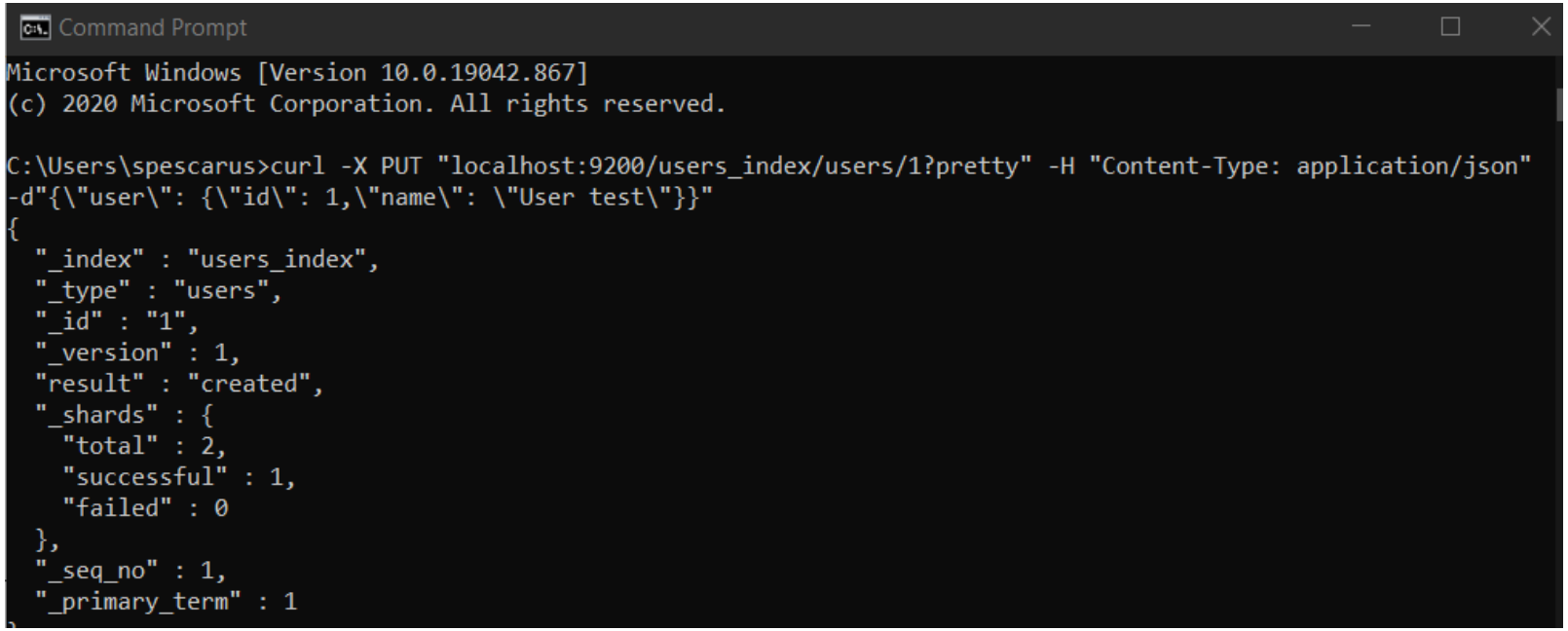
script support in filters and scorers

Configurable and Extensible

Many of Elasticsearch configurations can be changed while Elasticsearch is running, but some will require a restart (and in some cases reindexing). Most configurations can be changed using the REST API too.

Elasticsearch has several extension points - namely site plugins (let you serve static content from ES - like monitoring javascript apps), rivers (for feeding data into Elasticsearch), and plugins that let you add modules or components within Elasticsearch itself. This allows you to switch almost every part of Elasticsearch if so you choose, easily.

If you need to create additional REST endpoints to your Elasticsearch cluster, that is easily done as well.



Document Oriented

Store complex real world entities in Elasticsearch as structured JSON documents. All fields are indexed by default, and all the indices can be used in a single query, to return results at breath taking speed.

Schema free

Elasticsearch allows you to get started easily. Toss it a JSON document and it will try to detect the data structure, index the data and make it searchable.

Later, apply your domain specific knowledge of your data to customize how your data is indexed

Conflict management

Optimistic version control can be used where needed to ensure that data is never lost due to conflicting changes from multiple processes.

Active community

The community, other than creating nice tools and plugins, is very helpful and supporting.

Basic Concepts

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 elastic search which belongs to a cluster. Multiple nodes can be started on a single server for testing purposes, but usually you should have one node per server.

At startup, a node will use unicast(or multicast, if specified) to discover an existing cluster with the same cluster name and will try to join that cluster.

Index :
An index is like a 'database' in a relational database. It has a mapping which defines a type. An index is a logical namespace which maps to one or more primary shards and can have zero or more replica shards.

Type (Removed in version 7.0):
A type is like a 'table' in a relational database. A type has a list of fields that can be specified for documents of that type. The mapping defines how each field in the document is analyzed.

Document :

A document is a JSON document which is stored in elastic search. 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(also known in other languages as a hash /hashmap/ associative array) 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.

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. A field is similar to a column in a table in a relational database.

The mapping for each field has a field 'type' which indicates the type of data that can be stored in that field, ex. integer, string, object. The mapping also allows you to define(amongst other things) how the value for a field should be analyzed.

Mapping :
A mapping is like a 'schema definition' in a relational database. Each index has a mapping, which define search type within the index, plus a number of index-wide settings.
A mapping can either be defined explicitly, or it will be generated automatically when a document is indexed

Shard :
A shard is a single Lucene instance. It is a low-level "worker" unit which is managed automatically by elastic search. An index is a logical namespace which points to primary and replica shards.

Elasticsearch distributes shards amongst all nodes in the cluster, and can move shards automatically from one node to another in the case of node failure, or the addition of new nodes.

Primary Shard :
Each document is stored in a single primary shard. When you index a document, it is indexed first on the primary shard, then on all replicas of the primary shard.

By default, an index has 5 primary shards. You can specify fewer or more primary shards to scale the number of documents that your index can handle.

Replica Shard :
Each primary shard can have zero or more replicas. A replica is a copy of the primary shard, and hast wo purposes:

1. increase failover: a replica shard can be promoted to a primary shard if the primary fails.
2. increase performance: get and search requests can be handled by primary or replica shards.

Install Elastic Search

The latest version of elastic search can be downloaded from:
https://www.elastic.co/downloads/elasticsearch

1. Download and unzip Elastic Search

2. Run `bin/elasticsearch` (or `bin\elasticsearch.bat` on Windows)

3. Run `curl http://localhost:9200/, Invoke-RestMethod http://localhost:9200` with PowerShell or go to http://localhost:9200/ on browser

Elastic Search Monitoring

Elastic Search Head: https://github.com/mobz/elasticsearch-head

Chrome Extension: https://chrome.google.com/webstore/detail/elasticsearch-head/ffmkiejjmecolpfloofpjologoblkegm/

Is it running?

GET http://localhost:9200/


```
{
  "name" : "P5RO-PORT1551",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "HXeACs45RzSybQ4PiLYeEA",
  "version" : {
    "number" : "7.11.2",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "3e5a16cfec50876d20ea77b075070932c6464c7d",
    "build_date" : "2021-03-06T05:54:38.141101Z",
    "build_snapshot" : false,
    "lucene_version" : "8.7.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Let's play with Elastic Search

Indexing a document

```
curl -X PUT "localhost:9200/users_index/users/1?pretty" -H "Content-Type: application/json" -d"{\"user\": {\"id\": 1,\"name\": \"User test\"}}"
```

Response:

```
{
  "_index" : "users_index",
  "_type" : "users",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Get a document

curl -XGET http://localhost:9200/users_index/users/1?pretty

Response:


```
{
  "_index" : "users_index",
  "_type" : "users",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 1,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "user" : {
      "id" : 1,
      "name" : "User test"
    }
  }
}
```

Updating a document

```
curl -X PUT "localhost:9200/users_index/users/1?pretty" -H "Content-Type: application/json" -d"{\"user\": {\"id\": 1,\"name\": \"User test updated\"}}"

Response:
{
  "_index" : "users_index",
  "_type" : "users",
  "_id" : "1",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 2,
  "_primary_term" : 1
}
```

Searching

Searching and querying takes the format of: http://localhost:9200/[index]/[type]/[operation]

Search across all indexes and all types
http://localhost:9200/_search

Search across all types in the **users_index** index.
http://localhost:9200/users_index/_search

Search explicitly for documents of type users within the **users_index** index.
http://localhost:9200/users_index/users/_search

There's 3 different types of search queries
- Full Text Search (query string)
- Structured Search (filter)
- Analytics (aggregates)

Full Text Search (query string)

In this case you will be searching in bits of natural language for (partially) matching query strings. The Query DSL alternative for searching for "test" in all documents, would look like

Request: curl -XGET "localhost:9200/users_index/users/_search?pretty" -H "Content-Type: application/json" -d "{\"query\": {\"query_string\": {\"query\": \"test\" }}}"

Response:


```
{
 "took" : 1,
 "timed_out" : false,
 "_shards" : {
  "total" : 1,
  "successful" : 1,
  "skipped" : 0,
  "failed" : 0
 },
 "hits" : {
  "total" : {
   "value" : 1,
   "relation" : "eq"
  },
  "max_score" : 0.18232156,
  "hits" : [
   {
    "_index" : "users_index",
    "_type" : "users",
    "_id" : "1",
    "_score" : 0.18232156,
    "_source" : {
     "user" : {
      "id" : 1,
      "name" : "User test"
     }...
```

Structured Search (filter)

Structured search is about interrogating data that has inherent structure. Dates, times and numbers are all structured—they have a precise format that you can perform logical operations on. Common operations include comparing ranges of numbers or dates or determining which of two values is larger.

With structured search, the answer to your question is always a yes or no; something either belongs in the set or it does not. Structured search does not worry about document relevance or scoring—its imply includes or excludes documents.

Request:

```
curl -XGET "localhost:9200/users_index/users/_search?pretty" -H "Content-Type: application/json" -d "{\"query\": {\"bool\": {\"filter\": [{\"term\": {\"user.id\": \"1\"}}]}}}"
```

Analytics (aggregates)

Requests of this type will not return a list of matching documents, but a statistical break down of the documents.

Elasticsearch has functionality called aggregations, which allows you to generate sophisticated analytics over your data. It is similar to GROUP BY in SQL.

Request:

```
curl -XGET "http://localhost:9200/users_index/users/_search?pretty=true" -H "Content-Type: application/json" -d "{ \"aggs\": { \"all_users\": { \"value_count\": { \"field\" : \"user.id\" }}}}"
```

Response:

```
{
  "...
  },
  "hits" : {
    ...
    },
    "max_score" : 1.0,
    "hits" : [
      {
        ...
        }
      }
    ]
  },
  "aggregations" : {
    "all_users" : {
      "value" : 1
    }
  }
}
```

Elastic Search Limitations

Security : Elastic Search does not provide any build-in authentication or access control functionality.

Transactions : There is no much more support for transactions or processing on data manipulation.

Durability : Elastic Search is distributed and fairly stable, but backups and durability are not as high priority as in other datastores

Large Computations: Commands for searching data are not suited to "large" scans of data and advanced computation on the db side.

Data Availability : Elastic Search makes data available in "near real-time" which may require additional considerations in your application(ie: comments page where a user adds new comment, refreshing the page might not actually show the new post because the index is still updating).