trainologic

ELK

trainologic

ElasticSearch

- Started as a project named Compass by Shay Banon.
- Grew to become a multi-tenant and distributed search engine.
- Several other tools were built around it for the integrated ELK stack.

- Let’s go down memory lane…

trainologic

ElasticSearch

- ElasticSearch (ES) is a NoSQL storage engine (DB).
- Its main benefits are full-text-search and geo-indexing.
- One of the common use cases for using ES is log aggregation, however you can also use it in many other scenarios.

trainologic

ELK

- Because Logs analysis and aggregation became such an important use-case for ES, several other tools were developed to help with this popular use-case:
  - Kibana – a feature-rich front end for log analysis.
  - Logstash – log collector and pipe-line framework.
  - Beats – small and efficient log collectors.

trainologic

Basic Concepts

- Document – the object being indexed by ES. A document's format is always JSON.
- Index – A collection of documents. Its name must be all lowercase and unique.
- Type – Many changes here and deprecations (we'll discuss it shortly).
- Node – a member of the cluster.
- Cluster – a collection of nodes composing the ElasticSearch database.

trainologic

Type Deprecation

- Each document is assigned a single type (_*type* meta field).
- Combined with _*id* to generate the _*uid* field.
- In versions prior to 6.x, you could have specified multiple types in the same index (and search with multiple types).
- An analogy was made between index/types to database and tables.
- However, in the same index, same field name must have the same type.

trainologic

Type Alternatives

- You can use an index per type.
- You could alternately use custom fields.
- In ES 6.x, you can have a single type associated with an index.
- The _*uid* is now alias for _*id*.

trainologic

Indexing

- ElasticSearch is built on top of the awesome Lucene library.
- Provides the foundation for most indexing/search tools and frameworks in the Java ecosystem.
- At the heart of Lucene there is the **inverted index**.

- Let's understand it…

trainologic

Inverted Index

- Let's say that we have the following documents and their contents:

- Document **1**:
  - Hello, is anyone out there?
- Document **2**:
  - There is an exit. Outside.
- Document **3**:
  - One with the machine.

trainologic

Inverted Index

- A regular index is a document-to-term mapping.
- An inverted index maps the terms to their respective documents:
- E.g.:
  - hello – 1.
  - is – 1, 2.
  - one – 3.
  - anyone – 1.

trainologic

Mapping Data

- Before loading data into ES, we must create the mapping.
- This is because according to the mapping, text is processed.

- Not only on insert, but also on searches…

trainologic

String Types

- ES provides two types for indexing *string* fields:
  - text.
  - keyword.
- The *keyword* type provides for exact value search, whereas *text* types are fully analyzed.
- Use *keyword* for filtering, sorting and aggregations.
- Consider emails, hostnames, etc..

trainologic

Numeric Types

- Supported: *long, integer, short, byte, double, float, half_float, scaled_float.*
- All backed-by-floating-point types must have finite values and differentiate -0.0 from +0.0.
- All are unsigned.
- Java-based sizes.
- Scaled-float is more efficient if applicable.

trainologic

Boolean Types

- Starting from version 6.x, accepts JSON *true/false*, or string-values: “true”/”false”.
- Before that, ‘off’, ‘no’, ‘0’ and ‘’ were also accepted as false values.

trainologic

Date Type

- Dates are stored as *long* epoch representation of UTC value.
- The field itself can be:
  - String formatted date/time.
  - Long value of milliseconds since epoch.
  - Int value of seconds since epoch.

trainologic

Binary Type

- Accepts a Base64 value.
- Not stored or searchable by default.

trainologic

Range Types

- A range type allows for specifying a range value in documents.
- Can be:
  - Numeric: integer_range, float_range, long_range, double_range,
  - Date/time: date_range.
  - IP (CIDR): ip_range.

trainologic

The _source Field

- Enabled by default.
- Holds the original JSON document.
- Remember that even if this feature is enabled, the field is not indexed and can't be used for search.
- Incurs storage overhead.

- However, be careful before disabling this feature.
- Let's talk about it…

trainologic

The _source Field

- When you disable _*source*, you lose the following features:
  - Reindexing into a new index.
  - Reindex, update and update_by_query APIs.
  - On-the-fly highlighting.
  - Some debugging ability.

- Regarding disk storage, consider higher compression over disabling _*source*.

trainologic

The _source Field

- One of the classical use-cases for disabling *_source* is metrics collection.
- No need for highlighting, no need for updates or re-indexing (too fast data).

- Note that you should also probably disable the *_all* feature also.

trainologic

The _all Field

- Deprecated in 6.x.
- A special field that is the concatenation of all the fields in the document.
- Space is the delimiter.
- Note that the field is analyzed and indexed, but not stored!

trainologic

Kibana

- Kibana is the analytics and visualization tool in the ELK stack.
- Supports as wide variety of charts, dashboards and maps.

- Let’s start with the basics…

trainologic

Import Some Data

- Before loading some data into ES, we need to define the mapping.
- Let's see it for the example 'logs' dataset:

```
PUT /logstash-2015.05.18
{
  "mappings": {
    "log": {
      "properties": {
        "geo": {
          "properties": {
            "coordinates": {
              "type": "geo_point"
            }
          }
        }
      }
    }
  }
}
```

trainologic

Task

- Explore a few lines from the logs.jsonl file.
- Execute “*grep "logstash-" * | sort -u*"
- Define the additional mappings according to the index names.

trainologic

Bulk Insert

- For now, we won't go into the actual index/bulk-index APIs.
- Let's execute a bulk insert from the command line:

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

trainologic

Task

- Examine the results from the Kibana console using:
- GET _cat/indices?v

trainologic

cat API

- ElasticSearch provides comprehensive JSON-based REST API.
- However, JSON is a format suited for machines.
- You can use the **cat** API to get tabular human-readable representation.

- Let's go over the results of the previous command…

trainologic

Traffic Lights

- So, we see that the status is yellow for every index.
- In order to understand why, execute:

GET _cat/shards

- Now add the “?v” to the end of the URL.

- Can you figure the problem?

trainologic

Traffic Lights

- Red means that at least one primary shard was not assigned.
- Yellow means that for at least one shard, there are no replicas.
- Green means, well you know ☺

trainologic

Task

- For the ‘cat shards’ API, specify an index name before the ‘?’ character (to filter shards only for the specified index).
- Use a wildcard to display shards only for our *logs* data.
- Using the query parameter **h**, you can specify the columns to display as a comma-separated list.
- Please display: index, shard, state and unassigned.reason.
- Using the query parameter **s**, you can sort according to comma separated fields (you can specify ‘:asc’ or ‘:desc’ for every one).
- Please sort according to *index* and *shard*.

trainologic

Updating Index Setting

- Let's update an index to specify 'no-replicas':

```
PUT /logstash-2015.05.18/_settings
{
    "index" : {
        "number_of_replicas" : 0
    }
}
```

trainologic

Task

- Execute a single command to update the settings of all of our *logs* indices to have **0** replicas.
- Check the status of these indices.

- Make sure everything is green.

trainologic

Index Patterns

- The first thing you need to define when starting to work with Kibana is the index patterns!
- An index pattern specifies which indices in ES will be analyzed by Kibana.
- You can specify an exact index name, or you can use wildcards (‘*’).

trainologic

Task

- Specify an index pattern for our log data.
- Select *@timestamp* as the ‘time filter field name’.
- In the *Discover* tab, specify ‘Last 5 year’ (upper right corner).
- Zoom on the relevant section (containing data) in the time-series chart.

trainologic

Demo

- Filtering for *error* @tags.
- Filtering for security @tags.
- Doing it in a query (note ‘and’ vs. ‘AND’).
- Saving the query.

- View the query in the Management tab.

trainologic

Task

- Using UI operations, filter response status of 503.
- Filter geo.src from the US.
- Now, do that as a query.
- Change the query to filter all response codes representing errors (i.e., 4XX-5XX). Use range syntax (fieldname:[A to B]).
- Save your query.

trainologic

Task

- Insert the accounts.json into a *bank* index using:

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

- Update index settings to have a green status.
- Define the index pattern.
- Filter results having balance over 10000 (fieldname:>N) of females.
- Save your query.

trainologic

Highlighting

- Sometimes you don’t care about highlighting in the Discovery table.
- You can change that in the ‘Advanced Settings’ tab.
- Look for: “doc_table:highlight”.

trainologic

Visualization Theory

- A good visualization should communicate “the greatest number of ideas, in the shortest time, using the least amount of ink, in the smallest space” (Edward Tufte).

- Let’s see a master-piece…

trainologic

Visualization Demo

- Create a ‘vertical bar’ visualization for only females.
- Y-axis is average balance.
- X-axis is age bucketed at 10 years intervals.

- Saving the visualization.

trainologic

Task

- For the accounts data, create a vertical bar visualization of:
- Y-axis is avg. balance.
- Use split-chart by gender with sub-aggregation by age with interval of 5.

- Save the visualization.

trainologic

Task

- For the logs data, create a heat-map visualization over the ‘security errors’ query.
- Use geo.src for the X-Axis.

- Which country has most originating events?

trainologic

Task

- Create an area visualization for the *accounts* data.
- Count by State for the top 10 states.

- Create a pie visualization for the *logs* data.
- Slices should be host names.

- Create a coordinate-map for the *logs* data based on **geo.coordinates** field.

trainologic

Task

- Create a new index for *accounts*.
- Specify that the *address* field should be analyzed (hint: not a *keyword* type).

- Examine the results in the *Discover* tab.

- Try a search for partial addresses.

trainologic

Common Concepts

- Let’s go over some common ES search concepts…

trainologic

Scoring

- Search API calculates a score for each document.
- We'll go later over the fine details.

- Documents with higher score are returned first.

trainologic

Boosting

- Boosting alters the weights that fields get when computing the relevance score.
- Boosting can (and should) be specified on the query.
- It can also be specified in the mapping (index time).
- Index-time boosting is deprecated (since 5.x) and is generally a bad idea (can’t be changed without a reindex).

trainologic

Fuzziness

- Some queries support fuzziness.
- I.e., similarity matching.
- Calculation is based on Levenshtein Edit Distance.
- Measures the number of one character changes needed to transform one term into the other.

trainologic

Fuzziness Values

- The fuzziness parameter can be either set to an absolute value or to AUTO (the default).
- Auto fuzziness value depends on the term's length:
  - 0..2 – 0 (exact match).
  - 3..5 – 1.
  - > 5 – 2.

trainologic

Query & Filter Contexts

- There are two types of ‘contexts’ in the query DSL:
  - Query Context.
  - Filter Context.
- A query clause in a filter context will return a Boolean result regarding a document match check.
- A query clause in a query context will return a **score** representing how well a document matches the query clause.

trainologic

Analyzers

- Analyzers are responsible for the text processing flow of converting text into terms.
- Analyzers are actually composed of:
  - Character filters.
  - Tokenizers.
  - Token filters.

trainologic

Testing Analyzers

- The *analyze* API can be used to test the analyzers.
- Example:

Console

```
POST _analyze
{
  "tokenizer": "standard",
  "filter":  [ "lowercase", "asciifolding" ],
  "text":      "Is this déja vu?"
}
```

```
{
  "tokens": [
    {
      "token": "is",
      "start_offset": 0,
      "end_offset": 2,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
      "token": "this",
      "start_offset": 3,
      "end_offset": 7,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "deja",
      "start_offset": 8,
      "end_offset": 12,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "vu",
      "start_offset": 13,
      "end_offset": 15,
      "type": "<ALPHANUM>",
      "position": 3
    }
  ]
}
```

trainologic

Discussion

- Here you see an example using the *_analyze* API.
- Also note the response from the analyzer.
- It’s not only terms.
- You also get offsets and ordinal values!

trainologic

REST API

- The nodes in the cluster communicate with each other using the ‘transport interface’ (TCP-based).
- Nevertheless, the nodes also expose an HTTP-based REST API.

- Let’s see an example…

trainologic

REST API Examples

Console

```
1
2  GET _nodes
```


```
{
  "_nodes": {
    "total": 1,
    "successful": 1,
    "failed": 0
  },
  "cluster_name": "elasticsearch",
  "nodes": {
    "LgNEouTCQ9ScQCcby4MuDQ": {
      "name": "LgNEouT",
      "transport_address": "127.0.0.1:9300",
      "host": "127.0.0.1",
      "ip": "127.0.0.1",
      "version": "5.6.8",
      "build_hash": "688ecce",
      "total_indexing_buffer": 207775334,
      "roles": [
        "master",
        "data",
        "ingest"
      ],
      "settings": {
        "client": {
          "type": "node"
        },
        "cluster": {
          "name": "elasticsearch"
        },
        "http": {
          "type": {
            "default": "netty4"
          }
        },
        "node": {
          "name": "LgNEouT"
        },
```

trainologic

REST API Examples

Console

```
1
2 GET _cluster/health
```


```
1  {
2    "cluster_name": "elasticsearch",
3    "status": "yellow",
4    "timed_out": false,
5    "number_of_nodes": 1,
6    "number_of_data_nodes": 1,
7    "active_primary_shards": 1,
8    "active_shards": 1,
9    "relocating_shards": 0,
10   "initializing_shards": 0,
11   "unassigned_shards": 1,
12   "delayed_unassigned_shards": 0,
13   "number_of_pending_tasks": 0,
14   "number_of_in_flight_fetch": 0,
15   "task_max_waiting_in_queue_millis": 0,
16   "active_shards_percent_as_number": 50
17 }
```