POLITECNICO
MILANO 1863

Elasticsearch

Andrea Tocchetti
andrea.tocchetti@polimi.it

Elasticsearch - ELK Stack

Kibana

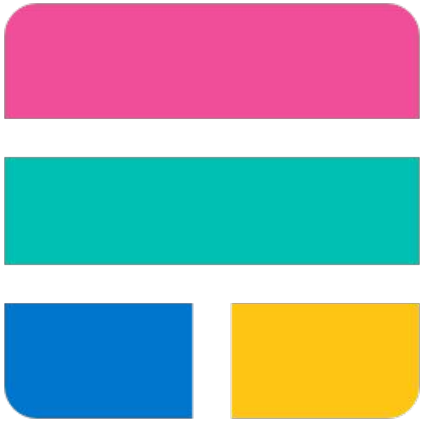
Visualize and Manage.

Elasticsearch

Store, Search, and Analyze.

Logstash + Beats

Aggregate, Process, and Ingest Data.



Elasticsearch - Introduction

Elasticsearch is the core of the Elastic Stack.

It's a Search and Analytic Engine that is

- Near real-time – Short latency,
- Full-text search,
- Distributed,
- JSON storage format,
- RESTful,
- Fast, Scalable, Resilient.

Elasticsearch - Indices

Index

- Data organization mechanism,
- Define the structure of documents via mappings,
- Partitioned in shards,
- Automatic indexing of new data.

Elasticsearch is based on the concept of **Inverted Index**.

Elasticsearch - Shards and Replicas

Shards distribute operations to

- Increase resistance to faults and hardware failures,
- Improve performances,
- Increase capacity to serve read requests.

There are two types of shards, namely **Primaries** and **Replicas**.

Each document in an index belongs to one **Primary Shard**.

Replica Shards are copies of Primary Shards stored on a different node.

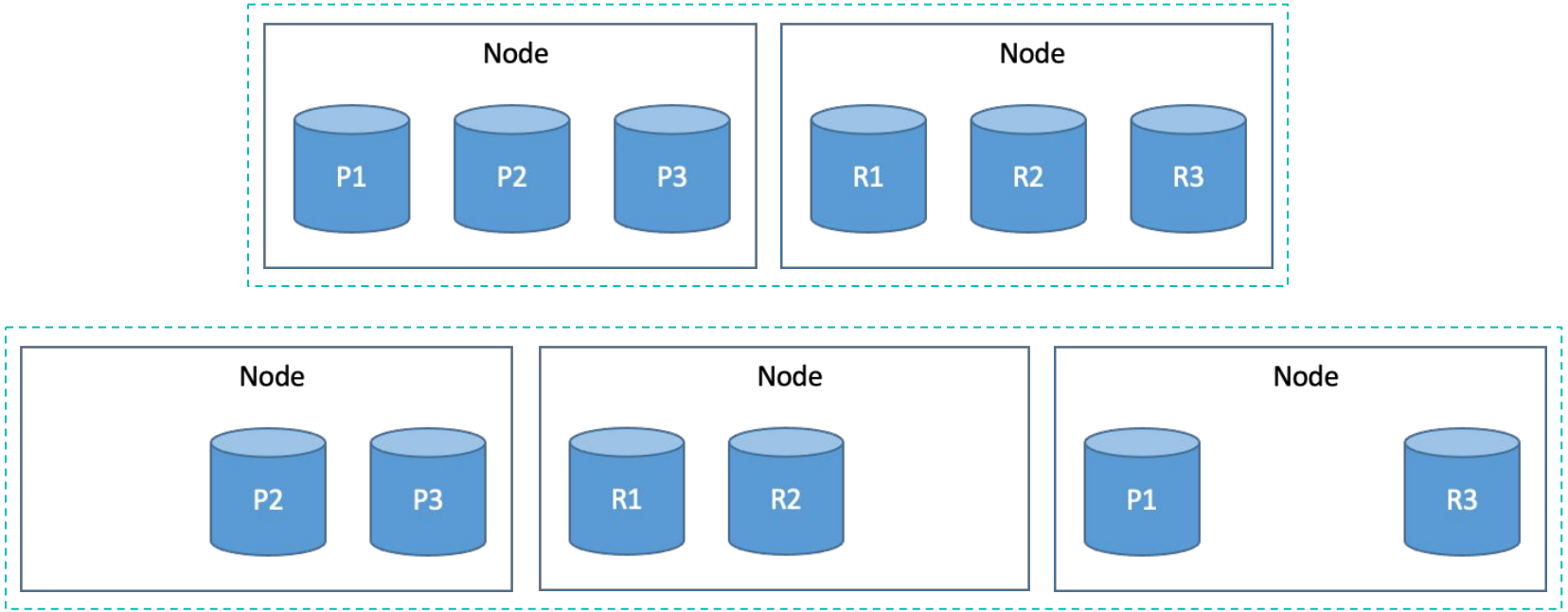
Elasticsearch - Shards and Replicas

Write Operation are performed on a Primary shard then on Replicas.
Read Operation are either performed on a Primary shard or a Replica.

The **number** of Primary Shards and the **size** of shards influence performances

- The more shards you have, the more the management overheads.
- The larger the shard size, the more it takes to move them (if needed).
- Querying a lot of small shards makes the processing per shard faster, but more queries means more overheads, consequently it may be more efficient to query a small amount of shards.

Elasticsearch - Shards and Replicas



Elasticsearch - Mapping

A document is described by a set of fields.

A **mapping** describes the data type of a document's fields.

There are two types of mapping

- **Dynamic Mapping** – Elasticsearch takes care of adding the new fields automatically when a document is indexed.
- **Explicit Mapping** – Makes you define the mapping. You must take care of any changes in the structure of the index manually.

Elasticsearch - Dynamic Mapping

When using the **Dynamic Mapping**, you must be aware of the possible drawbacks

- **Whenever** a new document is indexed, Elasticsearch adds its fields to the mapping, if they are not already present. Therefore, if a wide amount of **heterogeneous** documents are added, you may and up with a huge amount of fields which may end up breaking up the index.
- Elasticsearch is **Schemaless**, i.e., if no mapping is provided and dynamic mapping is applied, it will try to guess the structure of the document and the type of its fields.

N.B. you can’t change the mapping on an existing index that already has documents!

Elasticsearch - Interaction

Interactions with Elasticsearch happen through requests to REST endpoints.

The actions that can be performed depend on HTTP verb, namely

- **GET** is used to read document, indices metadata, mappings, etc.
- **POST** and **PUT** are often used to create new documents, indices, etc.
- **DELETE** is used to delete documents, indices, etc.

Requests can be sent from

- Command line
- Software packages (e.g., Postman)
- Developer tools (Kibana)

Elasticsearch - Interaction

Be aware of the differences between **POST** and **PUT**

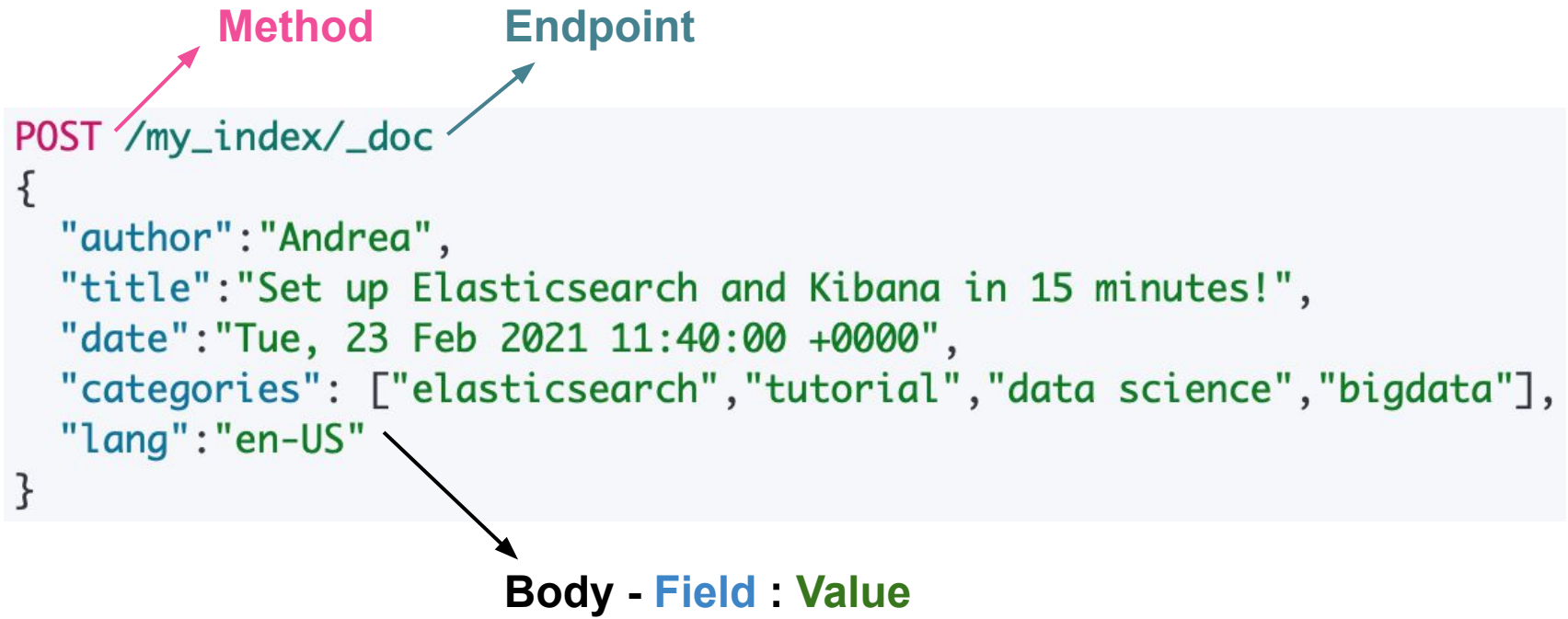
- **POST** doesn't require the ID of the resource
  - If omitted, Elasticsearch takes care of creating and assigning IDs to documents

```
POST /index_name/_doc
```

- **PUT** requires the ID of the resource

```
PUT /index_name/_doc/document_id
```

Elasticsearch - Interaction



Elasticsearch - Creating an Index

To create a new index, it's enough to use the **PUT** operator.

```
PUT /my_index
```

As an output, Elasticsearch will provide some metadata describing whether the index was successfully created.

```
"acknowledged": true,
"shards_acknowledged": true,
"index": "my_index"
```

Elasticsearch - Creating an Index

When defining an index, it is also possible to setup some parameters, namely one or more aliases, number of shards, number of replicas, and many more.

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

Elasticsearch - Explicit Mapping

When creating an index, it is possible to define the **mapping** straight away. Otherwise, it can still be defined after the index has been created.

```
PUT /my_index
{
  "mappings": {
    "properties": {
      "age": {"type": "integer"},
      "email": {"type": "keyword"},
      "name": {"type": "text"}
    }
  }
}
```

Elasticsearch - Add a Field to a Mapping

Given the rate at which the shape and complexity of the data evolves, sometimes it may be necessary to **add new fields** to an index.

```
PUT /my_index/_mapping
{
  "properties": {
    "surname": {"type": "text"},
    "index": "false"
  }
}
```

Index defines whether a field is indexed and queryable.

Elasticsearch - Updating a Mapping

Given the rate at which the shape and complexity of the data evolves, sometimes it may be necessary to **update the mapping** of some fields.

Be aware of the following when such updates are applied

- Updating the mapping of a field could invalidate the indexed data. If an update has to be made, it is necessary to **create** a new the index and **reindex** the data.
- Renaming a field could invalidate the indexed data. If a field must be renamed, **add** a new alias to that field instead.
- Each field has a set of mapping properties (e.g., index, and many more) which can be updated without any problem arising.

Elasticsearch - Visualising a Mapping

When Dynamic Mapping is applied, it is useful to **visualise the mapping** assigned by Elasticsearch to look for some flaws.

```
GET /my_index/_mapping
```

Sometimes, it is not even necessary to have a look at the whole mapping. Instead, it could be interesting to take a look at the mapping of a single field.

```
GET /my_index/_mapping/field/my_field
```

Elasticsearch - Field Types

binary	Binary value encoded as a Base64 string.
boolean	Either true or false.
keyword	Used for structured content (e.g., IDs, email addresses, hostnames, zip codes, etc.).
long	A signed 64-bit integer.
double	A double-precision 64-bit floating point number, restricted to finite values.
date	A string containing formatted dates. Internally managed as UTC.
object	A JSON object.
text	The traditional field type for full-text content. Text fields are **analyzed**.
geo_point	Latitude and longitude points.

Elasticsearch - Field Types

binary	Binary value encoded as a Base64 string.
boolean	Either true or false.
keyword	Used for structured content (e.g., IDs, email addresses, hostnames, zip codes, etc.).
long	A signed 64-bit integer.
double	A double-precision 64-bit floating point number, restricted to finite values.
date	A string containing formatted dates. Internally managed as UTC.
object	A JSON object.
text	The traditional field type for full-text content. Text fields are **analyzed**.
geo_point	Latitude and longitude points.

Elasticsearch - Analyzers

Standard Analyzer	It divides text into terms on word boundaries (using the UTS algorithm). It removes most punctuation, lowercases terms, and supports removing stop words.
Simple Analyzer	It divides text into terms whenever it encounters a character which is not a letter. It lowercases all terms.
Whitespace Analyzer	It divides text into terms whenever it encounters any whitespace character. It **does not** lowercase terms.
Stop Analyzer	It is like the **Simple Analyzer**, but also supports removal of stop words.
Pattern Analyzer	It uses a regular expression to split the text into terms. It supports lower-casing and stop words.
Language Analyzer	Language-specific analyzers (e.g., english, etc.)

Elasticsearch - Analyzers

If you are not sure about which analyzer is best suited for a specific case, they can be tested on sample **texts** to observe their behaviour. Moreover, if none of the pre-built analyzers is suited, **custom analyzers** can be built.

```
POST _analyze
{
  "analyzer": "standard",
  "text": "I really like sunsets."
}
```

Elasticsearch - Today's Dataset

From now on, all the queries will be performed on a small, sample dataset which contains data about video games on Steam (Download Link).

Field	Type
Developer	Keyword
Overview	Text
Publisher	Keyword
Tags	Text
Title	Text

Dynamic Mapping

Were they assigned the proper field type?

Elasticsearch - Retrieving a Document

Retrieving a single document can be done using its unique identifier.

```
GET /steam_overviews/_doc/0f5ERIIB_2yY7-gczBye
```

Elasticsearch returns a set of metadata (e.g., whether the document has been found, the name of the index, etc.) and a **source** field. Such a field contains all the data about the document that has been retrieved.

Elasticsearch - Retrieving a Document


```
"_source": {
  "overview": "EverQuest® II is the epitome of massively multiplayer gaming - the ultimate blend of deep
    features, heritage, and community.Explore an enormous online game where friends come together for
    adventure and community. Immerse yourself in a living, breathing fantasy world filled with exciting
    locales, mysterious lore, monsters, gods, and dragons. Vast, beautiful, and dangerous, EverQuest II
    sets the standard for MMORPG online gaming. EverQuest II is free to play.Your character. Your story.
    Your adventure. 110 levels of unparalleled gameplay  Strong heritage and lore spanning 14 years and 15
    expansions  A thriving, friendly community with thousands of guilds  Thousands of creatures to battle
    and quests to complete Hundreds of gorgeous, expansive, and dangerous environments to explore
    Independently leveled tradeskill system with deep quest lines  Full support of different play styles
    from, including Solo, Group, Raid, and PVP  Robust housing system with dozens of styles, thousands of
    items, ratings, and more  *Optional content available for purchase.",
  "publisher": "Daybreak Game Company ",
  "developer": "Daybreak Game Company ",
  "title": "EverQuest II",
  "tags": "['Free to Play', 'Massively Multiplayer', 'RPG', 'MMORPG', 'Fantasy', 'Crafting', 'Open World',
    'Adventure', 'Exploration', 'Multiplayer', 'Action', 'Character Customization', 'Singleplayer',
    'Classic', 'Sandbox', 'FPS']"
}
```

Elasticsearch - Queries

Elasticsearch supports two different categories of query clauses

- **Leaf Query Clauses** look for a particular value in a particular field. They can be used by themselves.
  - This category includes **match**, **term**, and **range** queries.
- **Compound Query Clauses** wrap other leaf or compound queries and are used to combine multiple queries in a logical fashion or to alter their behaviour.
  - This category includes **bool** queries and many more.

Depending on the context, these queries have different behaviours.

Elasticsearch - Query and Filtering Context

By default, Elasticsearch sorts matching search results by **Relevance Score**, which measures *how well each document matches a query*.

Each query type can calculate relevance scores differently. Score calculation also depends on whether the query clause is run in a **Query** or **Filter Context**.

Elasticsearch - Query and Filtering Context

Query Context → *"**How well** does this document match this query clause?"*

- Decides whether the document matches the query or not.
- The relevance score **IS** computed.
- Applied with the **query** parameter.

Filter Context → *"**Does** this document match this query clause?"*

- Decides whether the document matches the query or not.
- The relevance score **IS NOT** computed.
- Applied with the **filter** or **must_not** parameter.

Elasticsearch - Match Queries

Match Queries return documents that match a provided text, number, date or boolean value. Whenever a text field is provided, it is analyzed before matching.

Match Query

```
GET /steam_overviews/_search
{
  "query": {
    "match": {
      "publisher": {
        "query": "Daybreak Game Company "
      }
    }
  }
}
```

Elasticsearch - Match Queries

Match Queries can be performed using a compact style.

Match Query

```
GET /steam_overviews/_search
{
  "query": {
    "match": {
      "publisher": "Daybreak Game Company "
    }
  }
}
```

Elasticsearch - Match Queries

Match Queries are of type boolean. It means that the text provided is analyzed and the analysis process constructs a boolean query from the provided text.

Depending on the value of the **operator** parameter, the boolean query is either built using the **OR** operator or the **AND** operator.

- The **OR** operator matches the documents that contain at least 1 word among the ones returned by the analyzer.
- The **AND** operator matches the documents that contain all the words returned by the analyzer.

Elasticsearch - Match Queries

```
GET /steam_overviews/_search
{
  "query": {
    "match": {
      "overview": {
        "query": "free-to-play",
        "operator": "or"
      }
    }
  }
}
```

OR Query
free OR to OR play

AND Query
free AND to AND play

Elasticsearch - Match Queries

OR Query


```
"hits": {
  "total": {
    "value": 61,
    "relation": "eq"
  },
  "max_score": 1.4712923,
  "hits": [⟺]
}
```


AND Query


```
"hits": {
  "total": {
    "value": 35,
    "relation": "eq"
  },
  "max_score": 1.4712923,
  "hits": [⟺]
}
```

Elasticsearch - Term Queries

Term Queries return documents that contain an **exact term** in a provided field.

DO NOT use term queries for **text fields**. Remember that text fields are analyzed. Such an approach make it difficult to find exact matches.

The **value** parameter contains the term to look for in the document.

```
GET /steam_overviews/_search
{
  "query": {
    "term": {
      "publisher": {
        "value": "Ubisoft "
      }
    }
  }
}
```

Elasticsearch - Range Queries

Range Queries return documents that contain terms within a provided range. They are usually employed with numerical fields or dates.

Range Queries support the following operators

- **gt**, i.e., greater than
- **gte**, i.e., greater or equal than
- **lt**, i.e., less than
- **lte**, i.e., less or equal than

```
GET /steam_overviews/_search
{
  "query": {
    "range": {
      "publish_date": {
        "gte": "now-10d/d",
        "lte": "now",
        "boost": 2.0
      }
    }
  }
}
```

Elasticsearch - Term Queries

Some **query types** support the following parameters

- **boost** is a float number that multiplies the relevance score of a query. Useful for searches with multiple queries (default **1.0**).
- **case_insensitive** is a boolean that allows insensitive case matching (default **false**) in **term queries**.

These parameters must be added within the field, together with the **value** parameter.

Elasticsearch - Date Math

When performing ranged queries involving dates, it can be useful to know a little bit of the so-called **Date Math**. There are three main parts, namely

- An **anchor date** which is either now or a date followed by II.
- A **math expression** made of a math operator, a number and a time unit (e.g., d for days, M for months, Y for years, etc.).
- (optionally) An **operator** to round the date to the closest, chosen time unit.

e.g., now +1d /M, now -1Y /Y, 2022.07.28\I\I +2M /M

Elasticsearch - Boolean Queries

Boolean Queries match documents matching boolean combinations of other queries. It is built using one or more boolean clauses, each clause with a typed occurrence. The table below highlights the available boolean clauses.

must	The query must appear in matching documents and it will **contribute** to the score.
filter	The query must appear in matching documents. However, unlike **must** the score of the query will be **ignored**.
should	The query should appear in the matching document. If so, it will **contribute** to the score. If should is the only clause included, at least one of the conditions **must** be satisfied.
must_not	The query must not appear in the matching documents. The scoring is **ignored**.

Elasticsearch - Boolean Queries

Each of the **clauses** described may contain multiple conditions

```
GET /steam_overviews/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"tags": "Multiplayer"} },
        {"match": {"tags": "Action"} }
      ]
    }
  }
}
```

Elasticsearch - Boolean Queries

You can write Boolean Queries that include all or just a few of the **clauses** described.

```
GET /steam_overviews/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"tags": "Multiplayer"} }
      ],
      "filter": [
        {"term": {"publisher": "Ubisoft "} }
      ],
      "should": [
        {"match": {"tags": "Action"} }
      ]
    }
  }
}
```

Elasticsearch - Aggregations

Elasticsearch supports three different types of aggregations, namely

- **Metric Aggregations** calculate metrics from field values
- **Bucket Aggregations** group documents into buckets, based on field values, ranges, or other criteria.
- **Pipeline Aggregations** take input from other aggregations instead of documents or fields.

Aggregations return the **set of documents** before the aggregations themselves.

The **size** parameter defines how many of these documents are returned.

Elasticsearch - Aggregations

Aggregations can be run as part of a search by specifying the **aggs** parameter.

```
GET /steam_overviews/_search
{
  "size": 0,
  "aggs": {
    "games_per_publisher": {
      "terms": {
        "field": "publisher"
      }
    }
  }
}
```

```
"aggregations": {
  "games_per_publisher": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 44,
    "buckets": [...]
  }
}
```

```
{
  "key": "Daybreak Game Company ",
  "doc_count": 3
},
```

Elasticsearch - Aggregations

Aggregations can be combined with normal queries to filter the data and then perform the aggregation.

N.B. What would happen if we were to perform the **Aggregation** and then the **Term Query**?

Term Query

Aggregation

```
GET /steam_overviews/_search
{
  "size": 0,
  "query": {
    "term": {
      "publisher": {
        "value": "Ubisoft "
      }
    }
  },
  "aggs": {
    "games_per_publisher": {
      "terms": {
        "field": "publisher"
      }
    }
  }
}
```

Elasticsearch - Aggregations

Within the same aggs operator, it is possible to perform **multiple** aggregations on different fields.

```
GET /steam_overviews/_search
{
  "size": 0,
  "aggs": {
    "games_per_publisher": {
      "terms": {
        "field": "publisher"
      }
    },
    "games_per_developer": {
      "terms": {
        "field": "developer"
      }
    }
  }
}
```

```
"hits": {
  "total": {
    "value": 64,
    "relation": "eq"
  },
  "max_score": null,
  "hits": []
},
"aggregations": {
  "games_per_publisher": {⟷},
  "games_per_developer": {⟷}
}
```

Elasticsearch - Aggregations

Elasticsearch supports **Sub-Aggregations** which are computed for each aggregation by considering the documents in each one of them.

```
{
  "key": "Daybreak Game Company ",
  "doc_count": 3,
  "average_games_per_publisher": {
    "value": null
  }
},
```

```
GET /steam_overviews/_search
{
  "size": 0,
  "aggs": {
    "games_per_publisher": {
      "terms": {
        "field": "publisher"
      },
      "aggs": {
        "average_games_per_publisher": {
          "avg": {
            "field": "price"
          }
        }
      }
    }
  }
}
```

N.B. Price does not exists in the mapping, so it is set to null.


ANY
Questions?

Elasticsearch - Exercises

Define the **full mapping** describing a Person with 10 shards and 3 replicas.

```
PUT /people
{
  "settings": {
    "number_of_shards": 10,
    "number_of_replicas": 3
  },
  "mappings": {⟺}
}
```

```
"mappings": {
  "properties": {
    "personal_id": {
      "type": "keyword"
    },
    "name": {
      "type": "text"
    },
    "surname": {
      "type": "text"
    },
    "birth date": {
      "type": "date",
      "format": "yyyy-MM-dd"
    },
    "address": {
      "type": "text"
    },
    "eye_color": {
      "type": "keyword"
    },
    "height": {
      "type": "integer"
    }
  }
}
```

Elasticsearch - Exercises

Extract all the games which publisher is “Ubisoft”.

```
GET /steam_overviews/_search
{
  "query": {
    "term": {
      "publisher": "Ubisoft "
    }
  }
}
```

Elasticsearch - Exercises

Extract all games which tags field contains the “Free-to-Play” tag.

```
GET /steam_overviews/_search
{
  "query": {
    "match": {
      "tags": "Free-to-Play"
    }
  }
}
```

Elasticsearch - Exercises

Extract all games which tags field contains the “Free-to-Play” tag, prioritizing those whose developer is “Daybreak Game Company”.

```
GET /steam_overviews/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"tags": "Free-to-Play"} }
      ],
      "should": [
        {"term": {"developer": {"value": "Daybreak Game Company "} } }
      ]
    }
  }
}
```

Elasticsearch - Exercises

Extract all games which publisher is exactly “Lag Studios” without computing the relevance score.

```
GET /steam_overviews/_search
{
  "query": {
    "bool": {
      "filter": [
        {"term": {"publisher": "Lag Studios "} }
      ]
    }
  }
}
```

Elasticsearch - Exercises

For each developer, extract the count of the games which tags field contains the “Free-to-Play” tag.

```
GET /steam_overviews/_search
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {"match": {"tags": "Free-to-Play"} }
      ]
    }
  },
  "aggs": {
    "f2p_per_devs": {
      "terms": {
        "field": "developer"
      }
    }
  }
}
```