elasticsearch

ELASTICSEARCH

DATA PERSISTENCY

DERBOVEN MAXIM
IAO3

Inhoud

2 Wat is Elasticsearch

2.1 Inleiding

Elasticsearch kan voortdurend grote volumes informatie behandelen en automatisch schalen. Het is een nosql, json based data store. Je communiceert met elasticsearch met een RESTFUL API.

Relationeel Database	ElasticSearch
Database	Indexes
Tables	Patterns/Types
Rows	Documents
Columns	Fields

2.2 Hoe werk de Elastic stack (Inverted Index)

Inverted Indexes onderscheiden elastic search van andere search engines. Een inverted index mapt tekst op basis van zijn inhoud.

Om dit in een voorbeeld te verwerken nemen we even volgende 3 documenten:

1. Food is great
2. It is raining
3. Wind is strong

De index voor volgende documenten zou als volgt zijn

Term	**Frequency**	**Documents**
Food	1	1
Great	1	1
is	3	1,2,3

De termen worden alfabetisch opgeslagen zodat ze gemakkelijk gevonden kunnen worden.

Zoeken op meerdere termen is gedaan door een lookup te doen in de index. Het voert een UNION of INTERSECTION uit en haalt de relevante documenten op. Zulke indexes zijn verspreid over verschillende shards.

De ES index bestaat uit verschillende Lucene indexes, waarop deze weer bestaan uit index segmenten. De index files zijn niet aanpasbaar (behalve voor het verwijderen).

2.2.1 Analyzers

Analyseren is het proces om tekst te om te zetten in tokens of termen die worden toegevoegd aan de inverted index. Een analyzer bestaat uit volgende delen:

Character filter: Krijgt de originele tekst als stream en bewerkt deze door het toevoegen, verwijderen of veranderen van individuele characters.

Tokenizers: Krijgt een stream van charachter en breekt deze op in verschildende tokens

Token filters: krijgt de tokens en veranderd hier ook nog een laatste keer iets waar nodig.

2.2.2 Basis Concepten

Index

Collectie van JSON documents. Elk van deze documents zitten een index.

Shards

Omdat er geen limiet staat op het aantal documenten dat zich in een index kan bevinden, worden indexes vaak horizontaal geportioneerd over shards. Deze bevinden zich als nodes in een cluster.

Analyzer

Worden gebruikt zijdens het zoeken van documenten. Deze bevatten tokenizers die zinnen en tekst opsplitsen in tokens en token filters

Mapping

Combinatie van velden en analyzers Het bepaald hoe velden gestockeerd en geïndexeerd worden.

3 Opzetten van Elasticsearch & Kibana

3.1 Containers opzetten

Op de website staat duidelijk aangegeven hoe je het kan opzetten met docker.

1. Elasticsearch container opzetten met de volgende commands;

```
docker network create elastic
docker pull docker.elastic.co/elasticsearch/elasticsearch:8.4.3
docker run --name elasticsearch --net elastic -p 9200:9200 -p
9300:9300 -e "discovery.type=single-node" -t
docker.elastic.co/elasticsearch/elasticsearch:8.4.3
```

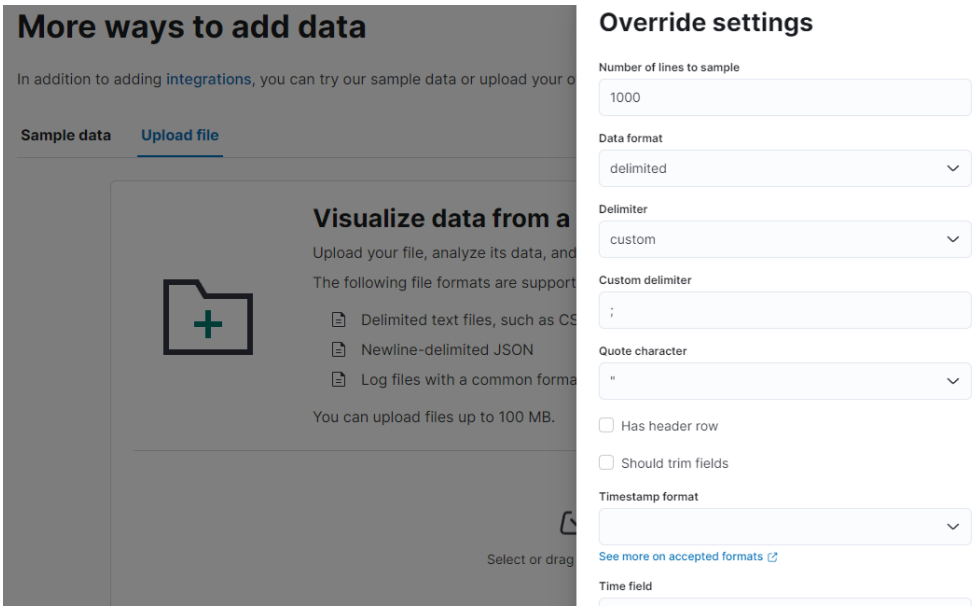
2. Start Kibana

```
docker pull docker.elastic.co/kibana/kibana:8.4.3
docker run --name kibana --net elastic -p 5601:5601
docker.elastic.co/kibana/kibana:8.4.3
```

3. Vervolgens naar de kibana webGUI surfen via localhost:5601. Daar moet je de enrollment token en password ingeven die in de terminal verscheen bij het instellen van de elasticsearch container.

3.2 Data inladen

Data inladen kan gemakkelijk via de kibana interface met een JSON of CSV.



Vervolgens geef je een index naam mee waarop je in de toekomst zal moeten zoeken.

4 Soorten Queries

Elasticsearch heeft een ruim aanbod aan zoekfunctionaliteiten. Ik bespreek hieronder de voornaamste en pas ze in het voorbeeld toe op onze hierboven gemaakte dataset.

4.1 Match Query

We gaan op zoek naar een bepaalde waarde in een bepaald veld met volgende query

```
{
  "query": {
    "match": {
      "Tweet": {
        "query" : "yes"
      }
    }
  }
}
```

Wanneer er bij query meerdere woorden voorkomen gaat hij automatisch de OR operator gebruiken. Om er voor te zorgen dat alle woorden moeten voorkomen kan dat met volgende query:

```
{
  "query": {
    "match": {
      "Tweet": {
        "query" : "yes no",
        "operator": "AND"
      }
    }
  }
}
```

Met "minimum_should_match" kunnen we bepalen hoeveel woorden uit de query er minstens moeten voorkomen in het veld.

```
{
  "query": {
    "match": {
      "Tweet": {
        "query" : "yes no",
        "minimum_should_match": 2
      }
    }
  }
}
```

4.2 Multi-Match Query

Met een multimatch query kunnen we de match functionaliteiten toevoegen om te zoeken op meerdere velden.

```
{
  "query": {
    "multi_match": {
        "query" : "Simon",
        "fields": ["Tweet","Name"]
    }
  }
}
```

4.3 Match Phrase

Met Match phrase kan je een volledige zin, inclusief de volgorde van de woorden matchen.

Optioneel: met de SLOP parameter kunnen we bepalen hoeveel de woorden mogen verwisselen van plaats om alsnog een match te vinden.

```
{
  "query": {
    "match_phrase": {
        "Tweet": {
        "query" : "This is a",
        "slop": 1
        }
    }
  }
}
```

4.4 Exists Query

Als we alle documenten willen zoeken die een bepaald veld bevatten kunnen we gebruiken maken van ‘exists’

```
{
    "query": {
        "exists": {
            "field": "Tweet"
        }
    }
}
```

En het tegenover gestelde:

```
{
    "query": {
        "bool": {
            "must_not": {
                "exists": {
                    "field": "Tweet"
                    }
            }
        }
    }
}
```

4.5 Range Query

Met een range query kun je bepaalde velden filteren op range. In het voorbeeld wordt er gezocht op een veld met een waarde tussen 5 en 10

```
{
    "query": {
        "range" : {
            "UserId" : {
                "gte" : 5,
                "lte" : 10
            }
        }
    }
}
```

4.5.1 Ranges (GTE, GT, LTE, LT)

GTE	Groter dan of gelijk aan (Greater Than or Equal to)
GT	Groter dan (Greater Than)
LTE	Kleinder dan of gelijk aan (Less Than or Equal to)
LT	Kleinder dan (Less Than)

4.6 Ids Query

Soms willen we documenten verkrijgen op basis van hun ID, dit kan gemakkelijk met volgende query

```
GET indexname/typename/documentId
```

Hiermee kunnen we echter maar 1 document met dat Id opvragen, waneer we op meerdere Ids willen matchen moeten we gebruik maken van volgende query.

```
{
    "query": {
        "ids" : {
            "values" : ["1", "4"]
        }
    }
}
```

4.7 Wildcard Query

Deze query staat toe om te zoeken op bepaalde velden met een wildcard.

```
{
    "query": {
        "wildcard": {
            "Tweet": {
                "value": "y*s"
            }
        }
    }
}
```

4.7.1 Wildcard Opties

*	Staat voor 0 of meer characters

4.8 Fuzzy

Een fuzzy query kan gebruikt worden om documenten terug te geven die hard lijken op de zoekterm. Dit kan gebruikt worden om spelling fouten recht te zetten.

```
{
  "query": {
    "fuzzy": {
      "Name": {
        "value": "Aarts Alexanded",
        "fuzziness": "AUTO"
      }
    }
  }
}
```

4.9 Filter Query

Elastic search haalt documenten op die aan de criteria voldoen en sorteert deze op basis van de score. Wanneer de score niet relevant is kunnen we gebruiken van een filter query en er de “bool” op toepassen. Deze geeft gewoon aan of hij matcht of niet.

```
{
  "query": {
    "bool": {
      "filter": [
        { "range": { "UserId": { "gte": "59995" }}}
      ]
    }
  }
}
```

4.10 Boosting

Om de beste resultaten vanboven te krijgen kunnen we elastic search een handje helpen door zelft te bepalen welke matches we belangrijker vinden en zo prioriteiten stellen.

```
{
  "query": {
    "multi_match": {
        "query" : "Simon",
        "fields": ["Tweet^3","Name"]
    }
  }
}
```

4.11 Sorting

4.11.1 Default sorting

Automatisch sorteert AS op “_score”. Dit is een score die aangeeft hoe goed een query matcht met de methodes van SA.

4.11.2 Sorting on field

Je kan ook de uitkomst op veld sorteren. In het voorbeeld sorteren we alles op naam

```
{
   "_source": ["UserId","Name","Tweet"],
  "sort": [
    {
      "Name": {
        "order": "ASC"
      }
    }
  ]

}
```

5 Toepassen op onze queries

5.1 Zoek de berichten van een bepaalde gebruikerId

Om te zoeken op gebruikerId maken we gebruik van de match query. Aangezien de score hier niet toe doet gebruiken we er een bool filter op.

```
{
  "query": {
    "bool": {
        "filter": [
            {
            "match": {
                "UserId": {
                    "query": "8",
                    "minimum_should_match": 1
                    }
                }
            }
        ]}
    }
}
```

5.2 Zoek de berichten op basis van zijn naam

Zorg ervoor dat je bij schrijffouten in zijn naam toch nog resultaten krijgt

Schrijffouten kunnen we opvangen met de fuzzy query. Omdat namen worden opgeslagen als keyword is het niet mogelijk daar een wildcard search op te doen en zullen we de volledige naam, al dan niet juist moeten typen.

```
{
  "query": {
    "match": {
      "Name": {
          "query": "Arts Dim",
          "fuzziness": "AUTO"
      }
    }
  }
}
```

5.3 Een zoekfunctie die zoekt in de Berichten

Maak hier zo goed mogelijk gebruik van de functionaliteiten van ElasticSearch

We gebruiken match_phrase met een slop van 2 om te matchen op de tweets. Wanneer onze data te groot wordt kunnen we het hiermee toch nog behouden. Vandaar gebruiken we geen wildcard of gewone match. We sorteren ook op _score om de beste resultaten bovenaan te krijgen.

```
{
    "sort": [
        {
            "_score": {
                "order": "DESC"
                }
            }],
    "query": {
        "match_phrase": {
            "Tweet": {
                "query" : "sad day",
                "slop": 10
            }
        }
    }
}
```