Elasticsearch Developer Cheat Sheet

Data Manipulation

Put/Get/Delete index

```
curl -XPUT localhost:9200/index-name -d '{"settings": { "number_of_shards": 1}}'

curl -XGET localhost:9200/index-name?pretty

curl -XDELETE localhost:9200/index-name
```

Put/Get/Delete template

```
curl -XPUT localhost:9200/_template/template-name -d '{
  "template": "logs*",
  "mappings": {
    "foo-type": {
      "properties": {
        "foo-field": {
          "type": "text"
        }
      }
    }
  },
  "settings": {
    "number_of_shards": 1
  }
}'

curl -XGET localhost:9200/_template/template-name?pretty

curl -XDELETE localhost:9200/_template/template-name
```

Bulk API

```
echo '{"index": { "_index": "logs01", "_type": "logs"}}
{"title": "this is an error"}
{"index": { "_index": "logs02", "_type": "logs"}}
{"title": "this is a warning"}
{"delete": { "_index": "logs03", "_type": "logs", "_id": "abc123"}}
' > /tmp/bulk
curl localhost:9200/_bulk?pretty --data-binary @/tmp/bulk
```

Ingest API (put/get/delete/simulate pipeline)

```
curl -XPUT localhost:9200/_ingest/pipeline/apache -d '{
  "description": "grok apache logs",
  "processors": [
    {
      "grok": {
        "field": "message",
        "patterns": ["%{COMBINEDAPACHELOG}%
{GREEDYDATA:additional_fields}"]
      }
    }
  ]
}'

curl -XGET localhost:9200/_ingest/pipeline/apache?pretty

curl -XDELETE localhost:9200/_ingest/pipeline/apache

curl -XPOSTlocalhost:9200/_ingest/pipeline/_simulate -d '{
  "pipeline": {
    "description": "grok apache logs",
    "processors": [
      {
        "grok": {
          "field": "message",
          "patterns": [
            "%{COMBINEDAPACHELOG}%{GREEDYDATA:additional_fields}"
          ]
        }
      }
    ]
  },
  "docs": [
    {
      "_source": {
        "message": "example.com - - [22/Apr/2016:18:52:51 +1200] \"GET
/images/photos/455.jpg HTTP/1.1\" 200 986 \"-\" \"Mozilla/5.0\" \"-\""
      }
    }
  ]
}'
```

Mapping Parameters

Field types

```
curl -XPUT localhost:9200/index-name -d '{
  "mappings": {
    "foo-type": {
      "properties": {
        "foo": {
          "type": "text"
        }
      }
    }
  }
}'
```

By default, string fields are mapped as both:

- **text** - full-text search
- **keyword** - exact search, sorting and aggregations

Numeric: **byte**, **short**, **integer**, **long**, **float**, **scaled_float**, **half_float**
Others: **boolean**, **ip**, **geo_point**, **geo_shape**

Analysis

Analyzer components:

- [character filters]
- tokenizer
- [token filters]

```
curl -XPUT localhost:9200/index-name -d '{
  "settings": {
    "analysis": {
      "char_filter": {
        "my_mapping_char_filter": {
          "type": "mapping",
          "mappings": ["& => and"]
        }
      },
      "analyzer": {
        "my_custom_analyzer": {
          "char_filter": ["my_mapping_char_filter"],
          "tokenizer": "whitespace",
          "filter": ["lowercase"]
        }
      }
    }
  },
  "mappings": {
    "foo-type": {
      "properties": {
        "foo": {
          "type": "text",
          "analyzer": "my_custom_analyzer"
        }
      }
    }
  }
}'
```

Analyze API:

```
curl -XPOST localhost:9200/index-name/_analyze -d '{
  "text": ["Fish & Chips"],
  "analyzer": "my_custom_analyzer"
}'
# reply
{
  "tokens": [
    {
      "token": "fish",
      "start_offset": 0,
      "end_offset": 4,
      "type": "word",
      "position": 0
    },
    {
      "token": "and",
...
    },
    {
      "token": "chips",
...
```

Important default analyzers:

- **standard** - tokenizes European languages OK, lowercases
- language (e.g. **english**, **dutch**) - selects the appropriate tokenizer (often **standard**), lowercases, removes stopwords and stems

Important character filters:

- **html_strip** - removes HTML elements and decodes HTML character entities
- **pattern_replace** - replaces regular expression matches

Important tokenizers:

- **standard** - the same used in the Standard Analyzer
- **letter** - tokens are only groups of letters
- **whitespace** - treats whitespaces as separators
- **pattern** - regular expression as separator
- **keyword** - treats the whole string as a token

Important token filters:

- **lowercase** or **uppercase** - folds cases
- **asciifolding** - folds non-ASCII characters to ASCII equivalents for european languages
- **stemmer** - reduces words to their roots (with configurable aggressiveness)
- **synonym** - adds synonym tokens to the index
- **ngram** - creates tokens out of groups of consecutive letters
- **edge ngram** - ngrams for prefixes
- **reverse** - flips character order (combine with edge ngram for suffix matching)
- **shingle** - word ngrams

Mapping options

```
curl -XPUT localhost:9200/index-name -d '{
  "mappings": {
    "foo-type": {
      "properties": {
        "foo": {
          "type": "text",
          "index_options": "docs",
          "norms": false,
          "fields": {
            "keyword": {
              "type": "keyword",
              "doc_values": true,
              "index": false
            }
          }
        }
      }
    }
  }
}'
```

- **doc_values** (true/false) - for sorting and aggregations on a field
- **index** (true/false) - for searching on a field
- **index_options** - whether to index only the term (**docs**), or also its frequency (**freqs**) and where it occurs (**positions** and **offsets**)
- **norms** (true/false) - for normalizing scores relative to field length
- **ignore_above** - don't index terms bigger than N characters

Queries

Full-text search

Lucene query syntax: **query_string**

```
curl localhost:9200/index-name/_search -d '{
  "query": {
    "query_string": {
      "query": "+fish +chips"
    }
  }
}'
```

Options:

- ***field:value*** to look in ***field***, or search in all fields (default) or in a specified **default_field**
- +requiredTerm -excludedTerm. Or you can say **requiredTerm1 AND requiredTerm2**
- **(firstName AND lastName) OR alias**
- **Ealsticsearch~1** (fuzziness of one character to tolerate typos)
- **"Sematext consulting Elasticsearch"~2** (slop of two words)
- **E?asticse***
- **date:[2017-01-01 TO 2018-01-01] OR rating:[3 TO *]**
- **boostThisTermByTen^10**
- escape special characters (?*~^:+-), use a backslash (\)

Text-box like search: **match**

```
  "query": {
    "match": {
      "foo": {
        "query": "bar baz",
        "operator": "OR"
      }
    }
  }
```

Options:

- **fuzziness** allows typos to be tolerated
- **cutoff_frequency** high-frequency terms are searched only on results of the low-frequency terms

For match on multiple fields: **multi_match**

```
    "multi_match": {
      "query": "fish chips",
      "fields": ["foo", "bar"]
    }
```

Can set **type** to:

- **best_fields** (default): takes the highest scoring field into account, optionally taking a fraction of the others (as defined by **tie_breaker**)
- **most_fields**: sums up scores of all fields (equivalent to best_fields with tie_breaker=1)
- **cross_fields**: treats multiple fields as one
- **phrase**: like best_fields, but matches phrases with a configurable **slop**
- **phrase_prefix**: like phrase, but considers the prefix of the last term

Filtering

Exact values: **term** and **terms**

```
    "term": {
      "foo": "fish"
    }
```

range

```
    "range": {
      "retweets": {
        "gte": 10,
        "lte": 20
      }
    }
```

Wrappers

Combining other queries: **bool**

```
    "bool": {
      "must": {
        "match": {
          "foo": "fish"
        }
      },
      "filter": {
        "range": {
          "retweets": {
            "gte": 10
          }
        }
      }
    }
```

Clauses:

- **must**: queries required both to produce a hit and for scoring
- **should**: queries that, if matched, contribute to the score
- **filter**: required queries, not influencing score (cacheable)
- **must_not**: cacheable queries that are required not to match

Alters score to [a subset of] results: **function_score**

```
    "function_score": {
      "query": {
        "match": {
          "foo": "fish"
        }
      },
      "functions": [
        {
          "filter": {
            "range": {
              "retweets": {
                "gte": 10
              }
            }
          },
          "weight": 5
        }
      ]
    }
```

Functions:

- **weight**/**random_score**: multiply the score by a static or a random number
- **field_value_factor**: multiply the score by a factor (e.g. square root) of the value of a field
- **linear**/**exp**/**gauss** decay: reduce the score based on how far the value of a field is from a specified **origin**
- **script**: use a script to generate a weight

Elasticsearch Developer Cheat Sheet

Aggregations

```
curl localhost:9200/index-name/_search -d '{
  "size": 0,
  "aggs": {
    "most_foos": {
      "terms": {
        "field": "foo.keyword"
      }
    }
  }
}'
```

Term occurrences

terms: by default, most occurrences of a term. Can order by other criteria (including other aggregations)
significant_terms: terms occurring more often in the query results compared to overall. More expensive, may want to use the **sampler** aggregation

Ranges

range: buckets of documents from defined numeric ranges
date_range/**ip_range**: same as range, but for dates and IPs
histogram/**date_histogram**: ranges are fixed from an interval

Statistics

```
  "aggs": {
    "avg_retweets": {
      "avg": {
        "field": "retweets"
      }
    }
  }
```

value_count/**min**/**max**/**avg**/**sum** of values from a field
percentiles from a numeric field are approximate
cardinality of terms is also approximate

Grouping by nesting aggregations

The following gets the top results, ordered by **_score**, grouped by the value of **bar** (one hit per value).

```
  "query": {
    "match": {
      "foo": "fish"
    }
  },
  "size": 0,
  "aggs": {
    "most_foo": {
      "terms": {
        "field": "bar.keyword",
        "order": {
          "max_score": "desc"
        }
      },
      "aggs": {
        "max_score": {
          "max": {
            "script": {
              "inline": "_score"
            }
          }
        },
        "top_hit": {
          "top_hits": {
            "size": 1
          }
        }
      }
    }
  }
```

Document Relationships

Objects

Good for one-to-one relations or when you're searching a single field:

```
curl -XPOST localhost:9200/blog/posts/ -d '{
  "title": "Fish & Chips",
  "author": {
    "first_name": "John",
    "last_name": "Smith"
  }
}'
```

Nested

When you need boundaries between objects (e.g. **first_name:jane AND last_name:smith**).
Mapping needs to specify that the parent field is **nested**:

```
  "mappings": {
    "posts": {
      "properties": {
        "authors": {
          "type": "nested"
        }
      }
    }
  }
```

Documents look like regular objects (even though they're separate Lucene documents):

```
  "authors": [
    {
    "first_name": "John",
    "last_name": "Smith"
    },
    {
      "first_name": "Jane",
      "last_name": "Adams"
    }
  ]
```

Queries (and aggregations) need to be aware of this and do the join:

```
  "query": {
    "nested": {
      "path": "authors",
      "query": {
        "match": {
          "authors.first_name": "Jane"
        }
      }
    }
  }
```

Parent-child

When updates are frequent and you want to avoid reindexing the whole ensemble (as you would with nested documents). These are completely separate documents, going in different types:

```
  "mappings": {
    "authors": {
      "_parent": {
        "type": "posts"
      }
    }
  }
```

Children point to parents via the **_parent** field:

```
curl -XPOST localhost:9200/blog/posts/1 -d '{
  "title": "Fish & Chips"
}'

curl -XPOST localhost:9200/blog/authors?parent=1 -d '{
  "first_name": "John",
  "last_name": "Smith"
}'

curl -XPOSTlocalhost:9200/blog/authors?parent=1 -d '{
  "first_name": "Jane",
  "last_name": "Adams"
}'
```

Like with nested documents, the query has to specify that a join needs to be done:

```
  "query": {
    "has_child": {
      "type": "authors",
      "query": {
        "match": {
          "first_name": "Jane"
        }
      }
    }
  }
```