Elasticsearch Cheat Sheet

A comprehensive cheat sheet for Elasticsearch, covering essential concepts, query syntax, API endpoints, and common operations.

Core Concepts & API Basics

Key Concepts

Index	A collection of documents with similar characteristics. Think of it as a database.
Document	A JSON document containing fields and their values. It's the basic unit of information.
Field	A key-value pair within a document. The key is the field name and the value is the data.
Mapping	Defines how a document and its fields are stored and indexed. Like a schema.
Shard	Indexes are divided into shards. Each shard is a fully-functional and independent "index" that can be hosted on any node in an Elasticsearch cluster.
Replica	A copy of a shard. Replicas provide redundancy and increase search capacity.

Basic API Endpoints

`PUT /<index_name>` - Create an index.

`GET /<index_name>` - Retrieve index information.

`DELETE /<index_name>` - Delete an index.

`POST /<index_name>/_doc` - Index a document. Elasticsearch will assign an ID.

`PUT /<index_name>/_doc/<_id>` - Index or update a document with a specific ID.

`GET /<index_name>/_doc/<_id>` - Retrieve a document by ID.

`POST /<index_name>/_search` - Search documents within an index.

Common HTTP Methods

GET	Retrieve information.
POST	Create a new resource or perform an action (e.g., search).
PUT	Create or update a resource at a specific ID. Replaces the entire document.
DELETE	Delete a resource.

Query DSL (Domain Specific Language)

Basic Query Structure

The Query DSL is based on JSON. The basic structure is:

```
{
  "query": {
    "<query_type>": {
      "<field_name>": {
        "<parameter>": "<value>"
      }
    }
  }
}
```

Match Query

`match` Analyzes the query and constructs a boolean query. Good for full-text search.

```
{
  "query": {
    "match": {
      "title": "quick brown fox"
    }
  }
}
```

`match_phrase` Matches exact phrases. The terms must be in the specified order.

```
{
  "query": {
    "match_phrase": {
      "message": "this is a test"
    }
  }
}
```

`match_all` Matches all documents. Useful for retrieving all documents in an index.

```
{
  "query": {
    "match_all": {}
  }
}
```

Term Query

`term` Finds documents that contain the *exact* term specified. Not analyzed.

```
{
  "query": {
    "term": {
      "user.id": "kimchy"
    }
  }
}
```

`terms` Finds documents that contain one or more of the *exact* terms specified.

```
{
  "query": {
    "terms": {
      "user.id": ["kimchy", "jordan"]
    }
  }
}
```

Boolean Query

`bool` A query that matches documents matching boolean combinations of other queries. Uses `must`, `should`, `must_not`, and `filter` clauses.

```
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title": "brown" }
}
      ],
      "filter": [
        { "term": { "tags": "search" } }
      ],
      "must_not": [
        { "range": { "date": { "gte":
"2024-01-01" } } }
      ],
      "should": [
        { "term": { "license": "pro" } }
      ],
      "minimum_should_match": 1
    }
  }
}
```

`must` The clause (query) must appear in matching documents and will contribute to the score.

`should` The clause (query) should appear in the matching document. If the `bool` query contains no `must` or `filter` clauses, then at least one `should` clause must match. Contributes to the score.

`must_not` The clause (query) must not appear in the matching documents. Is executed in filter context meaning that scoring is ignored and the clause is considered for caching.

`filter` The clause (query) must appear in matching documents. However unlike `must` the score of the query will be ignored. Filter clauses are executed in filter context, meaning that scoring is ignored and the clause is considered for caching.

Aggregations

Aggregation Basics

Aggregations allow you to compute statistics and analytics over your data. They are similar to SQL `GROUP BY`.

```
{
  "aggs": {
    "<aggregation_name>": {
      "<aggregation_type>": {
        "field": "<field_name>"
      }
    }
  }
}
```

You can nest aggregations.

Bucket Aggregations

`terms` Creates buckets based on unique terms in a field.

```
{
  "aggs": {
    "popular_tags": {
      "terms": {
        "field": "tags.keyword",
        "size": 10
      }
    }
  }
}
```

`date_histogram` Creates buckets based on date intervals.

```
{
  "aggs": {
    "articles_per_month": {
      "date_histogram": {
        "field": "publish_date",
        "calendar_interval": "month",
        "format": "yyyy-MM-dd"
      }
    }
  }
}
```

`range` Creates buckets based on numeric or date ranges.

```
{
  "aggs": {
    "price_ranges": {
      "range": {
        "field": "price",
        "ranges": [
          { "to": 50 },
          { "from": 50, "to": 100 },
          { "from": 100 }
        ]
      }
    }
  }
}
```

Metric Aggregations

`avg` Calculates the average of a numeric field.

```
{
  "aggs": {
    "avg_price": {
      "avg": {
        "field": "price"
      }
    }
  }
}
```

`sum` Calculates the sum of a numeric field.

```
{
  "aggs": {
    "total_sales": {
      "sum": {
        "field": "sales"
      }
    }
  }
}
```

`min` Calculates the minimum value of a numeric field.

```
{
  "aggs": {
    "min_price": {
      "min": {
        "field": "price"
      }
    }
  }
}
```

`max` Calculates the maximum value of a numeric field.

```
{
  "aggs": {
    "max_price": {
      "max": {
        "field": "price"
      }
    }
  }
}
```

`cardinality` Calculates the approximate number of unique values in a field. Useful for counting distinct users.

```
{
  "aggs": {
    "distinct_users": {
      "cardinality": {
        "field": "user_id"
      }
    }
  }
}
```

Mappings & Settings

Mapping Types

`text`	Used for full-text search. Analyzed into individual terms.
`keyword`	Used for exact-value matching, filtering, and sorting. Not analyzed.
`date`	Stores dates. Can be formatted. `"format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis"`
`integer`, `long`, `float`, `double`	Numeric types.
`boolean`	Stores boolean values (true/false).
`object`	Used for nested JSON objects.
`nested`	Used for arrays of JSON objects. Allows querying each object in the array independently.

Explicit Mapping

You can define the mapping explicitly when creating an index.

```
PUT /my_index
{
  "mappings": {
    "properties": {
      "title": { "type": "text" },
      "publish_date": { "type": "date",
"format": "yyyy-MM-dd" },
      "author_id": { "type": "keyword" }
    }
  }
}
```

If no mapping is defined, Elasticsearch will attempt to infer the mapping dynamically (Dynamic Mapping).

Index Settings

`number_of_shards`	The number of primary shards an index should have. Defaults to 1 in newer versions. Can only be set at index creation.
`number_of_replicas`	The number of replica shards each primary shard should have. Defaults to 1. Can be changed dynamically after index creation. `PUT /my_index/_settings { "number_of_replicas": 2 }`
`analysis`	Configures analyzers, tokenizers, token filters, and character filters for text analysis. Allows for customizing how text is indexed and searched.