# Der Elastic Stack

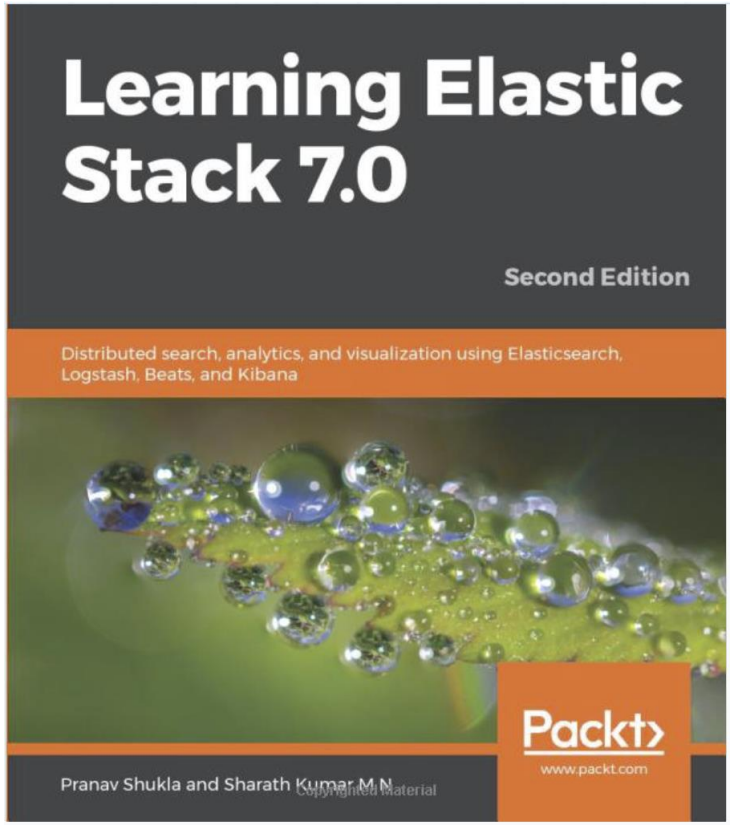
Prof. Dr. Alexander Paar

Duale Hochschule Schleswig-Holstein

# Literatur

Learning Elastic Stack 7.0 (Packt) von Pranav Shukla und Sharath Kumar M N

…



# Motivation

Enorme Datenmengen durch mobile Anwendungen, Blogs, soziale Netzwerke

Aufgabe: Suche in den Daten, Analyse der Daten

Der ELK-Stack

- Speicherung, Suche und Analyse mit Elasticsearch
- Visualisierung mit Kibana
- Laden und Transformation der Daten mit Logstash und Beats

# Definition

Elasticsearch ist eine horizontal skalierende Software für echtzeitfähige verteilte Suche und Datenanalyse für eine Vielzahl von Anwendungsfällen

Elasticsearch basiert auf Lucene, einer Programmbibliothek zur Volltextsuche. Lucene ist freie Software und ein Projekt der Apache Software Foundation

Beispiel: Wikipedia verwendet Lucene, seit 2014 via Elasticsearch

# Eigenschaften von Elasticsearch

Dokumentenorientierung

Schemafreiheit

Suche

Analyse

Gute Konnektivität

Einfache Bedienbarkeit

Einfacher Betrieb

Echtzeitfähigkeit

Schnelligkeit

Fehlertoleranz

# Dokumenten-orientierung & Schemafreiheit

Elasticsearch ist dokumentenorientiert und speichert JSON-Dokumente

Ein JSON-Dokument entspricht einem Record in einer relationalen Datenbank, aber…

Elasticsearch ist schemafrei

Die JSON-Dokumente können unterschiedlich strukturiert sein

```
{
  "name": "John Smith",
  "address": "121 Sunset Blvd., Beverly Hills, 90210",
  "age": 42
}
```

```
{
  "name": "John Doe",
  "age": 23,
  "email": "john.doe@example.com"
}
```

# Suche

Volltextsuche (auch: Volltextrecherche) ist das Auffinden von Wörtern oder Wortgruppen in einer Vielzahl gleicher oder verschiedenartiger Dateien

Google-Suche: Für eine Suchanfrage werden die *besten* Treffer sortiert angezeigt

Beispiel: „heavenly touches“, „heaven touches“, „heavenly touch“

Vergleiche: SQL-Anfragen mit `WHERE`, `EQUALS (=)` oder `LIKE`

Die Inhalte aller Dateien müssen indiziert werden

Elasticsearch sucht in Text, Zahlen, Datumsangaben, Geodaten, IP-Adressen, …

Who will believe my verse in time to come,
If it were fill’d with your most high deserts?
Though yet, heaven knows, it is but as a tomb
Which hides your life and shows not half your parts.
If I could write the beauty of your eyes
And in fresh numbers number all your graces,
The age to come would say “This poet lies;
Such heavenly touches ne’er touch’d earthly faces.”
So should my papers, yellowed with their age,
Be scorn’d like old men of less truth than tongue,
And your true rights be term’d a poet’s rage
And stretched metre of an antique song:
But were some child of yours alive that time,
You should live twice, in it and in my rhyme.

Sonnet 17, William Shakespeare

# Analyse

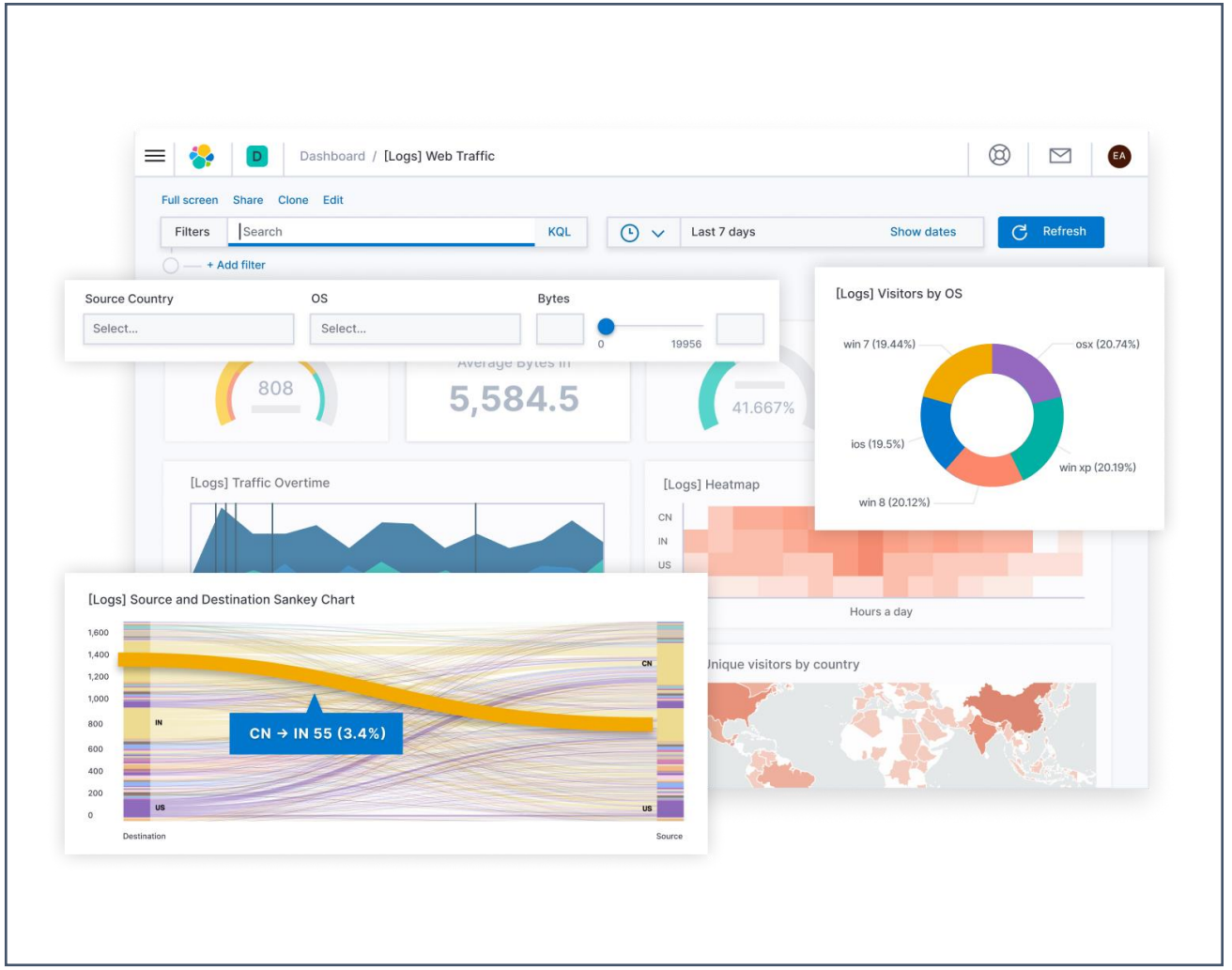
Elasticsearch ist auch ein Werkzeug für die Analyse von Daten

Suche: Zoome in die Daten und finde ausgewählte Dokumente für die Suchanfrage

Analyse: Zoome heraus und erkenne die großen Zusammenhänge

Beispiel: Wie viele Besucher einer Webseite verwenden welches Betriebssystem? Aus welchen Ländern kommen die Besucher?

Elasticsearch ermöglicht die Aggregation von Daten mit verschiedenen Operatoren



# Die weiteren Eigenschaften

Client-Bibliotheken für Elasticsearch gibt es für Java, C#, Python, JavaScript, …

Elasticsearch bietet eine umfassende REST API

Elasticsearch kann auf einem oder auf hunderten Knoten betrieben werden

Knoten können einfach hinzugefügt werden

Daten sind für Anfragen bereits kurz nach dem Indizieren verfügbar

Elasticsearch kann tausende Dokumente pro Sekunde indizieren

Elasticsearch indiziert *alle* Felder eines Dokuments

Elasticsearch ist fehlertolerant durch (dynamische) Redundanz

# Die Komponenten des ELK Stack

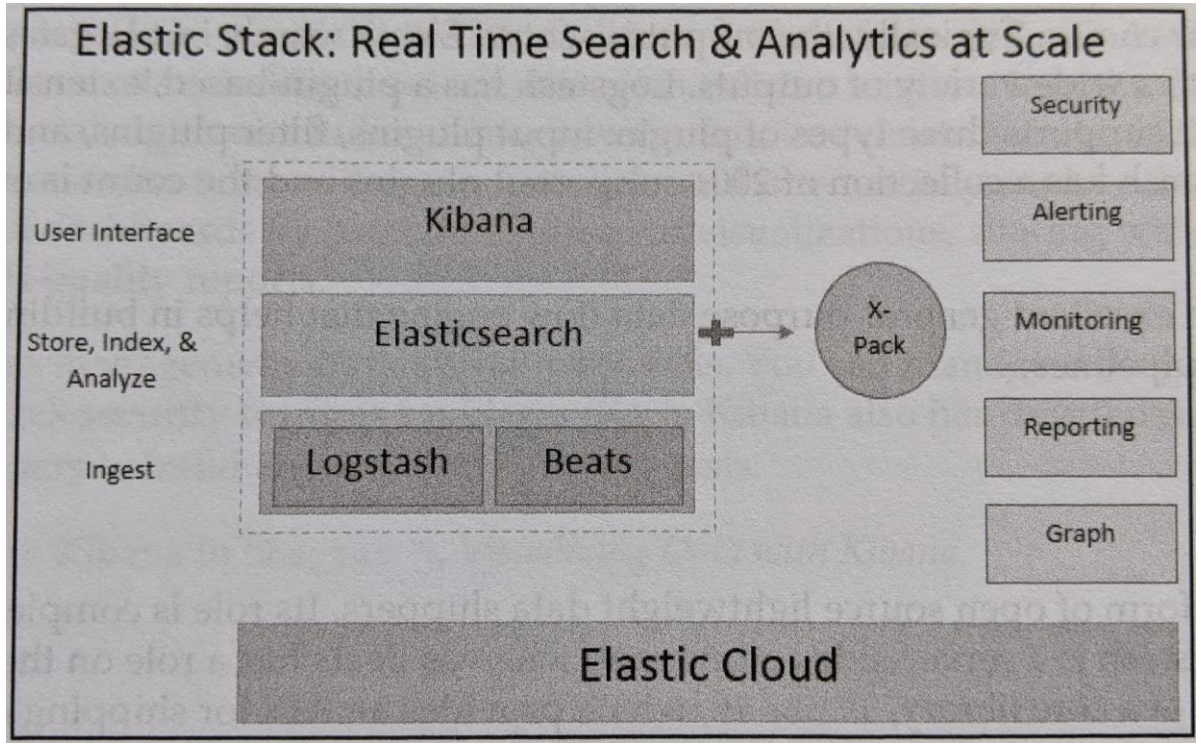
Elasticsearch speichert die Daten und ermöglicht die Suche und Analyse

Logstash zentralisiert die Transformation und das Laden von Daten

- Logstash ist serverbasiert
- Logstash ist mit Plug-Ins erweiterbar

Beats ist eine Client-basierte Plattform für Data Shippers

- Bibliothek `libbeat` für Datenübertragung und Logging
- Filebeat lädt Log-Dateien in einen zentralen Logstash-Server



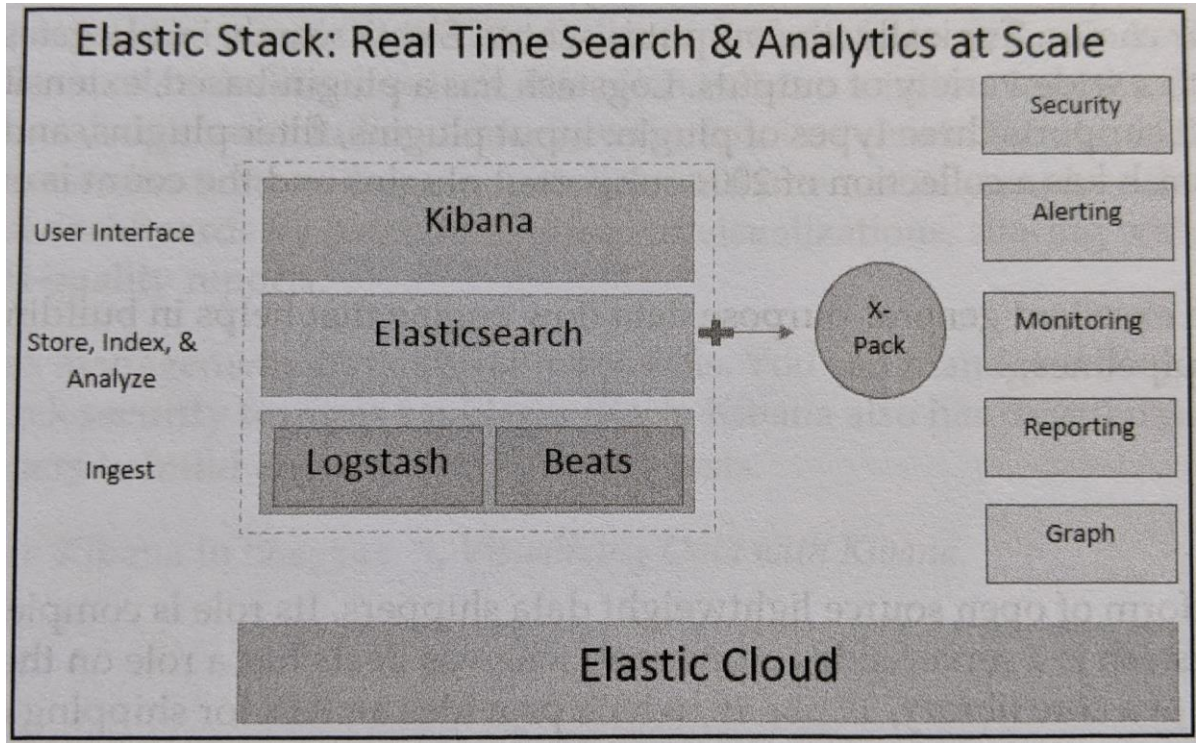
# Die Komponenten des ELK Stack

Kibana ist das Visualisierungs-Tool

- Das „Fenster“ in den Elastic Stack
- Development Tools für das Erstellen und Testen von REST API-Anfragen

Das X-Pack enthält Funktionalität für „production ready” Elastic-Installationen

- Security: Authentifizierung und Autorisierung
- Monitoring von Elasticsearch-Clustern und Kibana
- Reporting von Kibana-Visualisierungen
- Alerting via Email, Slack, …
- Graph für das Explorieren von Beziehungen in Elasticsearch-Daten (mit Kibana UI)
- Ausreißererkennung in Zeitreihendaten mit maschinellem Lernen



# Anwendungsfälle

Log- und Sicherheitsanalyse

- Weiterleitung von Logdaten mit Filebeats
- Zentrale Transformation mit Logstash
- Indizierung, Suche und Analyse mit Elasticsearch
- Visualisierung von Fehlern, Warnungen, usw. mit Kibana
- Dashboards für Echtzeitmonitoring mit Kibana

Produktsuche

- Relevanz-basierte Volltextsuche in Produktdatenbanken

Analyse von Zeitreihen- und Geodaten

- „Slicing and dicing“
- Kombination mit Kartendaten

Web-Suche

- Apache Nutch als Webcrawler und Elasticsearch als Suchmaschine

# Installation von Elasticsearch

1. Download Elasticsearch
2. Entpacken in ein Programmverzeichnis
3. Test der Installation


```
https://www.elastic.co/downloads/elasticsearch

bin/elasticsearch.bat

curl http://localhost:9200
```


```
{
  "name" : "85d46c54906f",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "ct0cQWY5Sia-Bggw8dh2yQ",
  "version" : {
    "number" : "7.8.1",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "b5ca9c58fb664ca8bf9e4057fc229b3396bf3a89",
    "build_date" : "2020-07-21T16:40:44.668009Z",
    "build_snapshot" : false,
    "lucene_version" : "8.5.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

# Installation von Kibana

1. Download Kibana
2. Entpacken in ein Programmverzeichnis
3. Test der Installation

kibana

```
https://www.elastic.co/downloads/kibana

bin/kibana.bat

http://localhost:5601
```

# Elasticsearch

Konzepte

# Kibana Dev Tools Console

Die Kibana Dev Tools Console ist ein Frontend für die Elasticsearch REST API

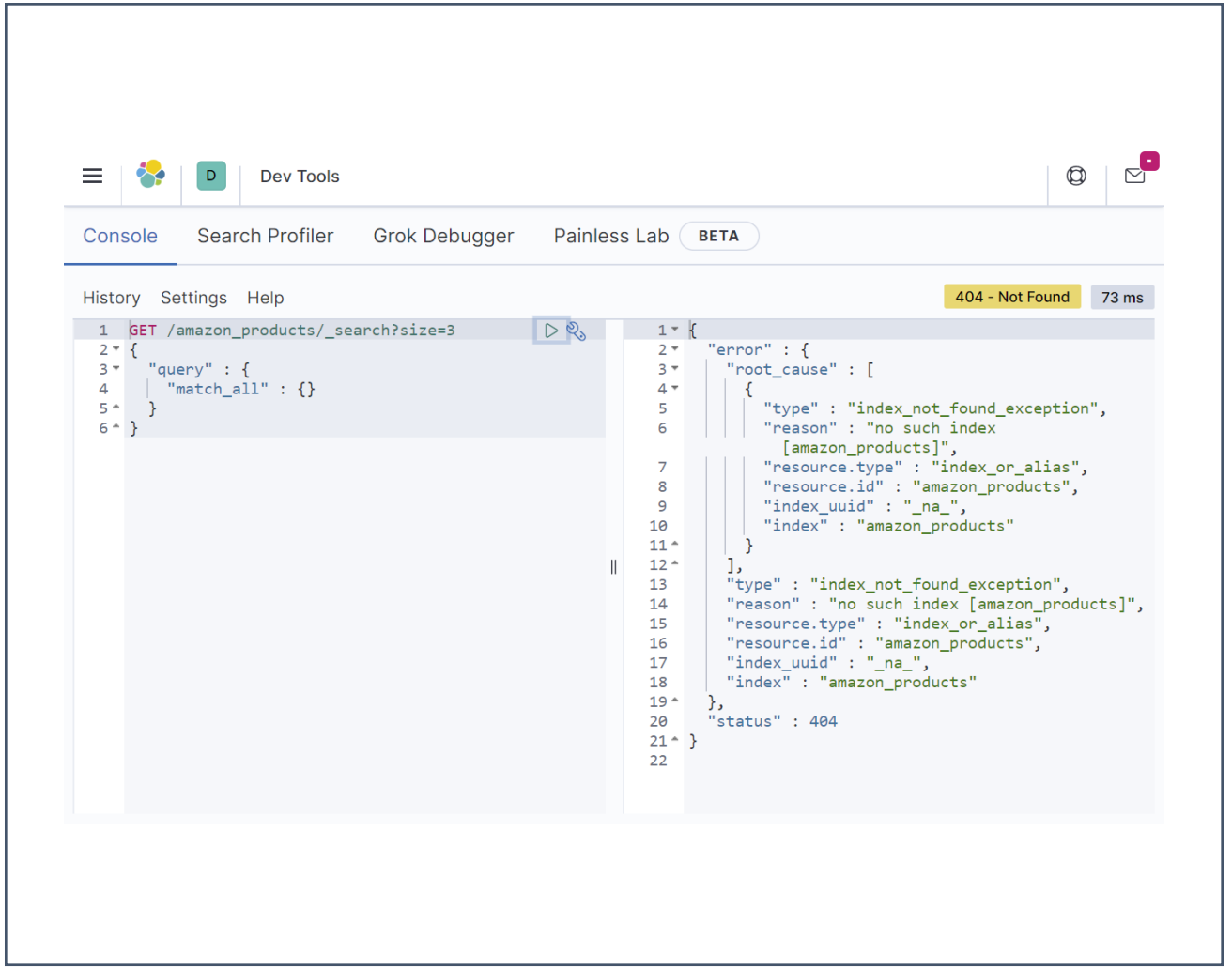
```
GET /
```

anstatt

```
curl http://localhost:9200
```

**Elasticsearch REST APIs**:

https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html



# Was ist HTTP

Hypertext Transfer Protocol

Zustandsloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht

HTTP Requests & Responses

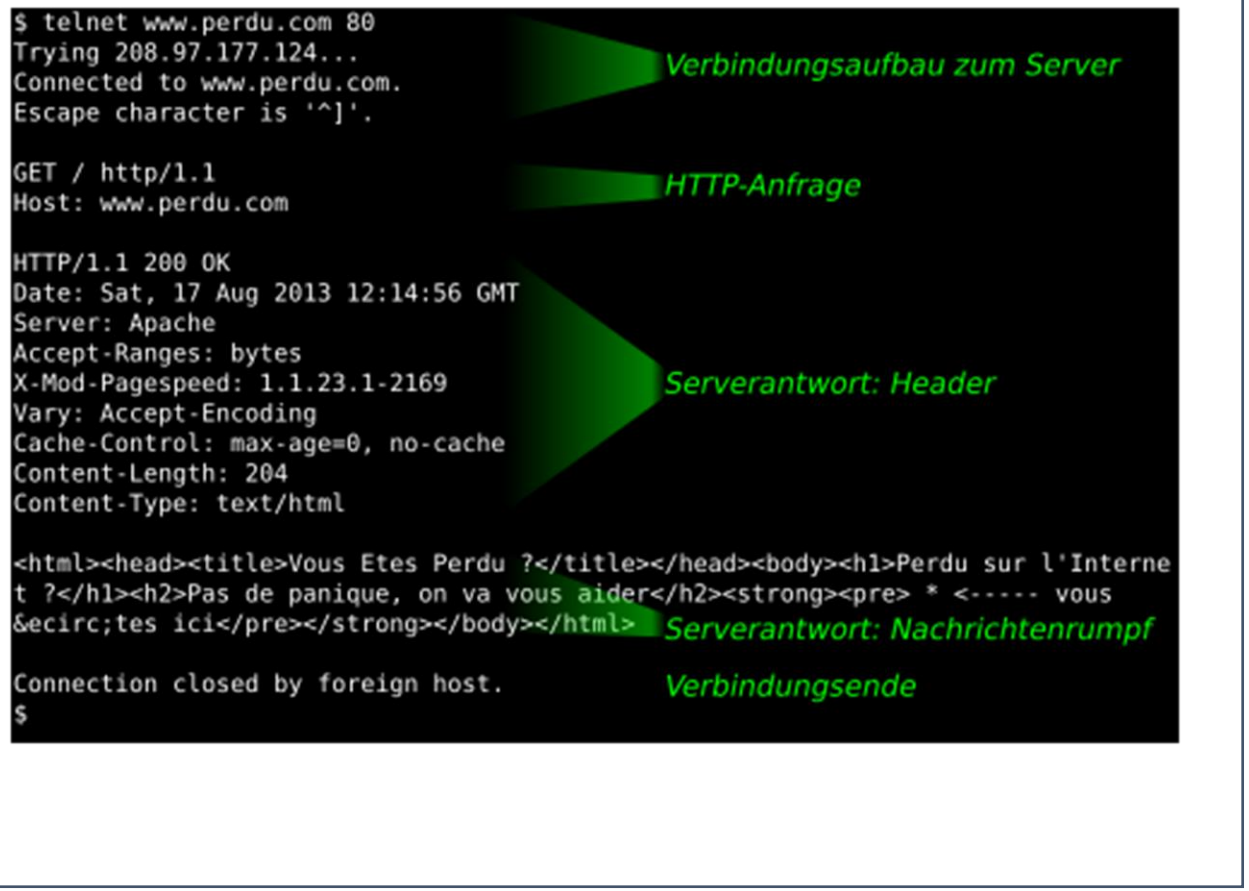
- Ein Server erhält Anfragen und sendet Antworten mit Statuscode zurück

HTTP Header

- Menge von Schlüssel-Werte-Paaren für zum Beispiel den „Content-Type“ oder ein Authentifizierungstoken

HTTP Body

- Daten die gesendet oder empfangen werden



# HTTP Requests

## GET

- Fordert die angegebene Ressource vom Server an

## POST

- Fügt eine neue (Sub-)Ressource unterhalb der angegebenen Ressource ein

## PUT

- Die angegebene Ressource wird angelegt. Wenn die Ressource bereits existiert, wird sie geändert.

## DELETE

- Löscht die angegebene Ressource.

```
GET     http://example.com/api/users     // Get all users
GET     http://example.com/api/users/1   // Get single user 1
POST    http://example.com/api/users     // Add user
PUT     http://example.com/api/users/1   // Update user 1
DELETE  http://example.com/api/users/1   // Delete user 1
```

# HTTP-Statuscodes

Ein HTTP-Statuscode wird von einem Server auf jede HTTP-Anfrage als Antwort geliefert

- Mitteilung, ob die Anfrage erfolgreich bearbeitet wurde

Die erste Ziffer eines Statuscodes stellt die Statusklasse dar

- Definiert in einigen RFCs
- Manchmal auch proprietäre Codes für Status- und Fehlermeldungen

https://de.wikipedia.org/wiki/HTTP-Statuscode

1xx – Informationen
2xx – Erfolgreiche Operation
- 200 OK
- 201 Created
- 204 No Content

3xx – Umleitung
- Not Modified

4xx – Client-Fehler
- 400 Bad Request
- 401 Unauthorized
- 404 Not Found

5xx – Server-Fehler
- 500 Internal Server Error
- 503 Service Unavailable

9xx – Proprietäre Fehler

# Konzepte von Elasticsearch

In einem RDBMS gibt es Tabellen, Zeilen, Spalten, Schemata

Elasticsearch ist dokumentenorientiert

Dokumente sind innerhalb von Indizes und Dokumenttypen organisiert

Abstraktionen in Elasticsearch

- Indizes
- Dokumenttypen
- Dokumente
- Cluster
- Knoten
- Shards und Replikate
- Abbildungen und Datentypen
- Invertierte Indizes

```
GET /_cat/indices


PUT /books/_doc/1
{
  "title": "The Art of Computer Programming",
  "author": "Donald E. Knuth",
  "isbn": "978-0321751041",
  "pages": 3168,
  "price": 155.24
}
```

snippets/concepts/indexes.txt

# Indizes

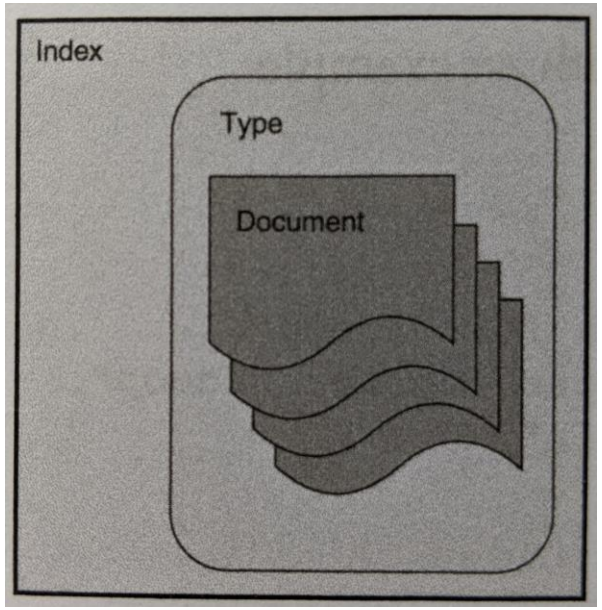
Ein Index ist ein logischer Container für Dokumente eines Dokumenttypes

Ein Index muss seit Elasticsearch 6.0 genau einen einzigen Dokumenttyp enthalten

Konfigurationsparamter können für Indizes und für Dokumenttypen definiert werden

Analogie zu einem RDBMS:

| **Elasticseach** | **RDBMS** |
|---|---|
| Index | Datenbankschema |
| Dokumenttyp | Tabelle |
| Dokument | Zeile in einer Tabelle |



# Dokumenttypen

Elasticsearch ist schemafrei

Innerhalb eines Dokumenttyps können beliebig strukturierte JSON-Dokumente gespeichert werden

Seit Elasticsearch 8.0 gibt es keine Dokumenttypen („document types“) mehr!

Aber: In der Praxis sollten sehr unterschiedlichstrukturierte Dokumente nicht innerhalb des gleichen Dokumenttyps organisiert sein

Besser: Speicherung unterschiedlich strukturierter Dokumente in eigenen Indizes

```json
{
  "title": "The Art of Computer Programming",
  "author": "Donald E. Knuth",
  "isbn": "978-0321751041",
  "pages": 3168,
  "price": 155.24
}
```

```json
{
  "firstName": "Donald",
  "middleName": "Ervin",
  "lastName": "Knuth",
  "birthday": "1938-01-10",
  "nationality": "American",
  "awards": 22,
}
```

# Dokumente

Ein JSON-Dokument ist eine Menge von Schlüssel-Werte-Paaren / Feldern

Jedes Feld hat seinen eigenen Datentyp

Analogie zu einem RDBMS:

| **Elasticseach** | **RDBMS** |
|---|---|
| Schlüsselname | Name eines Attributs |
| Wert | Wert eines Attributs |

Interne Metafelder

- `_id`: Identifier eines Dokuments, kann selbst vergeben oder automatisch generiert werden
- `_type`: Dokumenttyp
- `_index`: Name des Index

```
{
  "title": "The Art of Computer Programming",
  "author": "Donald E. Knuth",
  "isbn": "978-0321751041",
  "pages": 3168,
  "price": 155.24
}
```

# Knoten

Elasticsearch ist ein verteiltes System aus mehreren Prozessen auf mehreren Rechnern in einem Netzwerk

Ein Knoten (engl. node) in Elasticsearch ist ein einzelner Server, d.h. eine Instanz eines Elasticsearch-Prozesses

Dieser Server kann Teil eines Clusters sein und trägt dort zur Indizierung, Suche und anderen Operationen bei

Jeder Knoten erhält beim Start eine ID, außerdem kann ein Name vergeben werden (`node.name` in `config/elasticsearch.yml`)

Die Konfigurationsdateien der Knoten sind im YAML-Format

# Cluster

Ein Elasticsearch Cluster umfasst einen oder mehrere Knoten

Jeder Knoten ist automatisch Teil eines Clusters (evtl. mit nur einem einzigen Knoten)

Standardmäßig versucht ein Knoten Teil eines „Elasticsearch“-Clusters zu werden

Der Name des Clusters kann konfiguriert werden (`cluster.name` in `config/elasticserach.yml`)

Mit der Konfiguration `discovery.type=single-node` bildet ein Knoten einen Single Node Cluster

# Shards

Ein Shard ist ein Ausschnitt eines Index

Mit Shards können Indizes auf mehrere Knoten verteilt werden

Sharding ist die Aufteilung eines Index in mehrere Shards

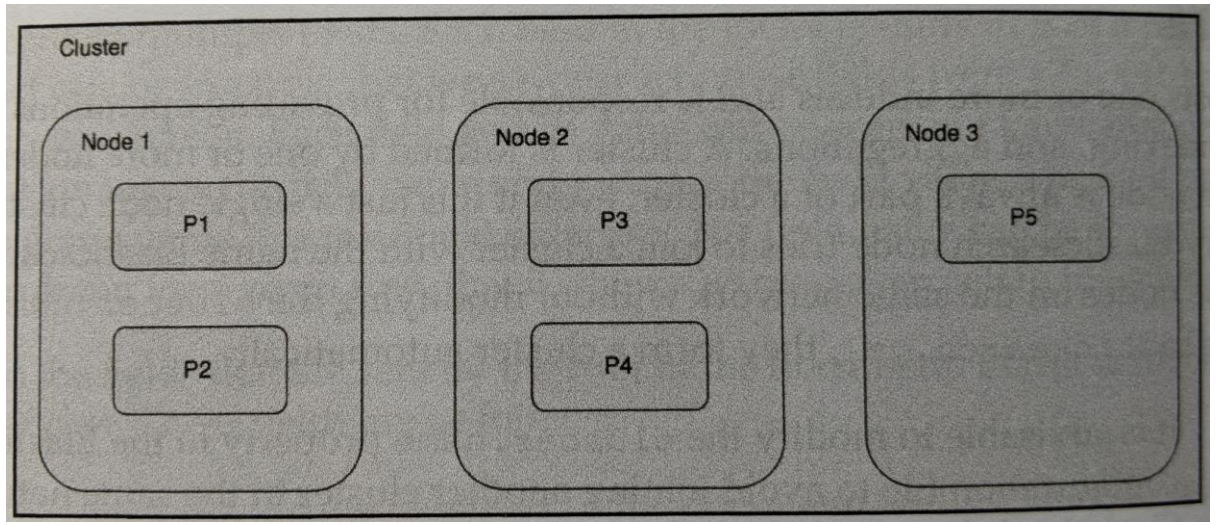
Standardmäßig besteht ein Index aus 5 Shards

Nach der Erzeugung eines Index kann die Anzahl Primary Shards nicht mehr geändert werden

Für die Bearbeitung von Anfragen an einen Index berücksichtigt Elasticsearch automatisch alle Shards dieses Index

**cat shards API**:

https://www.elastic.co/guide/en/elasticsearch/reference/current/cat-shards.html

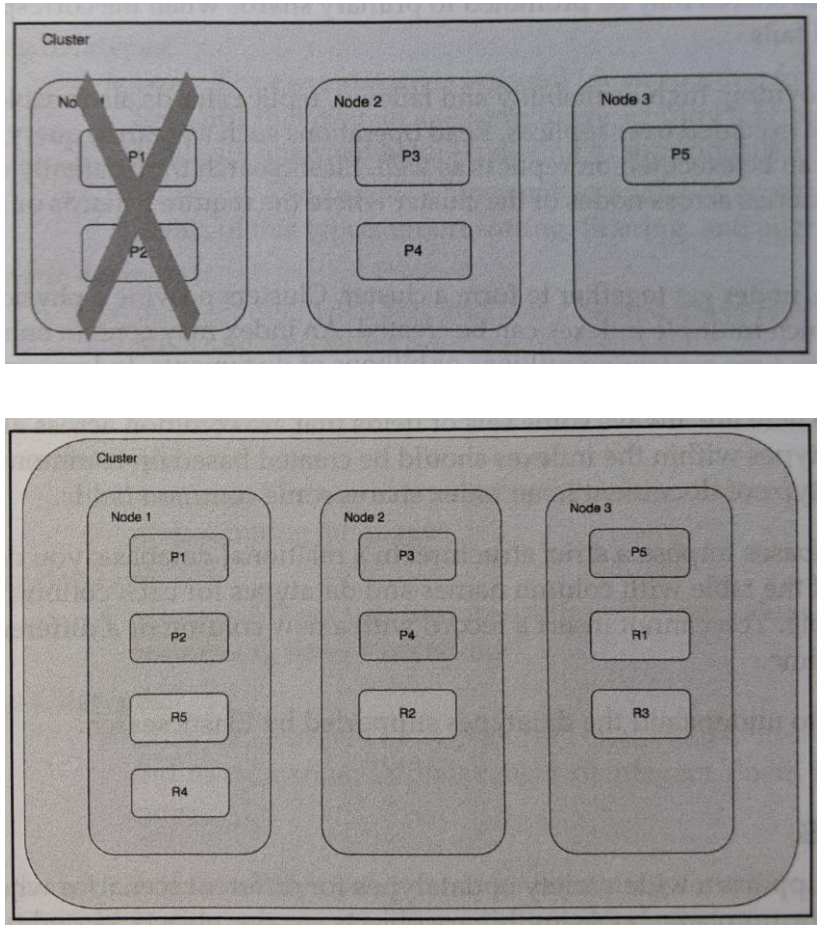


# Replikate

Verteilte Systeme wie Elasticsearch sollen auch bei Ausfällen weiter funktionieren

Replikate (engl. replica shards) bieten dynamische Redundanz für Datensicherheit und Performanz durch Lastverteilung

Elasticsearch verteilt Anfragen automatisch an die benötigten Shards und Replikate



# Datentypen

Elasticsearch ist schemafrei, aber in der Praxis sind die Daten es meistens nicht

Für JSON-Dokumente mit gemeinsamen Feldern sollte ein Index definiert werden

In einem RDBMS müssten alle Attribute bei der Definition einer Tabelle feststehen

Wie in einem RDBMS und in Programmier-sprachen gibt es in Elasticsearch Datentypen

Für jedes Feld eines JSON-Dokuments ist ein Datentyp definiert

Weitere spezielle Datentypen: IP, Geo-Point, Geo-Shape

Außerdem: Arrays, Object, Nested, Join

### Basisdatentypen

- `boolean`: Boolesche Werte
- `binary`: Binäre Werte als Base64-Zeichenkette

### String-Datentypen

- `text`: Analysierter unstrukturierter Text
- `keyword`: Strukturierter Text für Datenanalyse

### Numerische Datentypen

- `byte`, `short`, `integer`, `long`: Vorzeichenbehaftete Ganzzahlen
- `float`, `double`: IEEE 754-Fließkommazahlen
- `half_float`: 16-bit IEEE 754-Fließkommazahlen
- `scaled_float`: Fließkommazahl aus `long` und `double`-Faktor

### Datumstypen

- `date`: Datumsangaben als Zeichenkette oder Unixzeit

### Bereichsdatentypen

- `integer_range`, …: Bereiche von Zahlen, Datumsangaben, …

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html

# Abbildungen

Ein Index mit Abbildungen (engl. mappings) von Feldern nach Datentypen kann mit der **Create index API** definiert werden

Ein Index der noch nicht existiert wird automatisch beim Hinzufügen des ersten Dokuments erzeugt

- Elasticsearch inferiert automatisch die Datentypen (engl. dynamic mapping)
- https://www.elastic.co/guide/en/elasticsearch/reference/current/dynamic-mapping.html

**Get mapping API**:

https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-get-mapping.html

```
PUT /persons/_doc/1
{
  "firstName": "Donald",
  "middleName": "Ervin",
  "lastName": "Knuth",
  "birthday": "1938-01-10",
  "nationality": "American",
   "awards": 22
}


GET /persons/_mapping
```

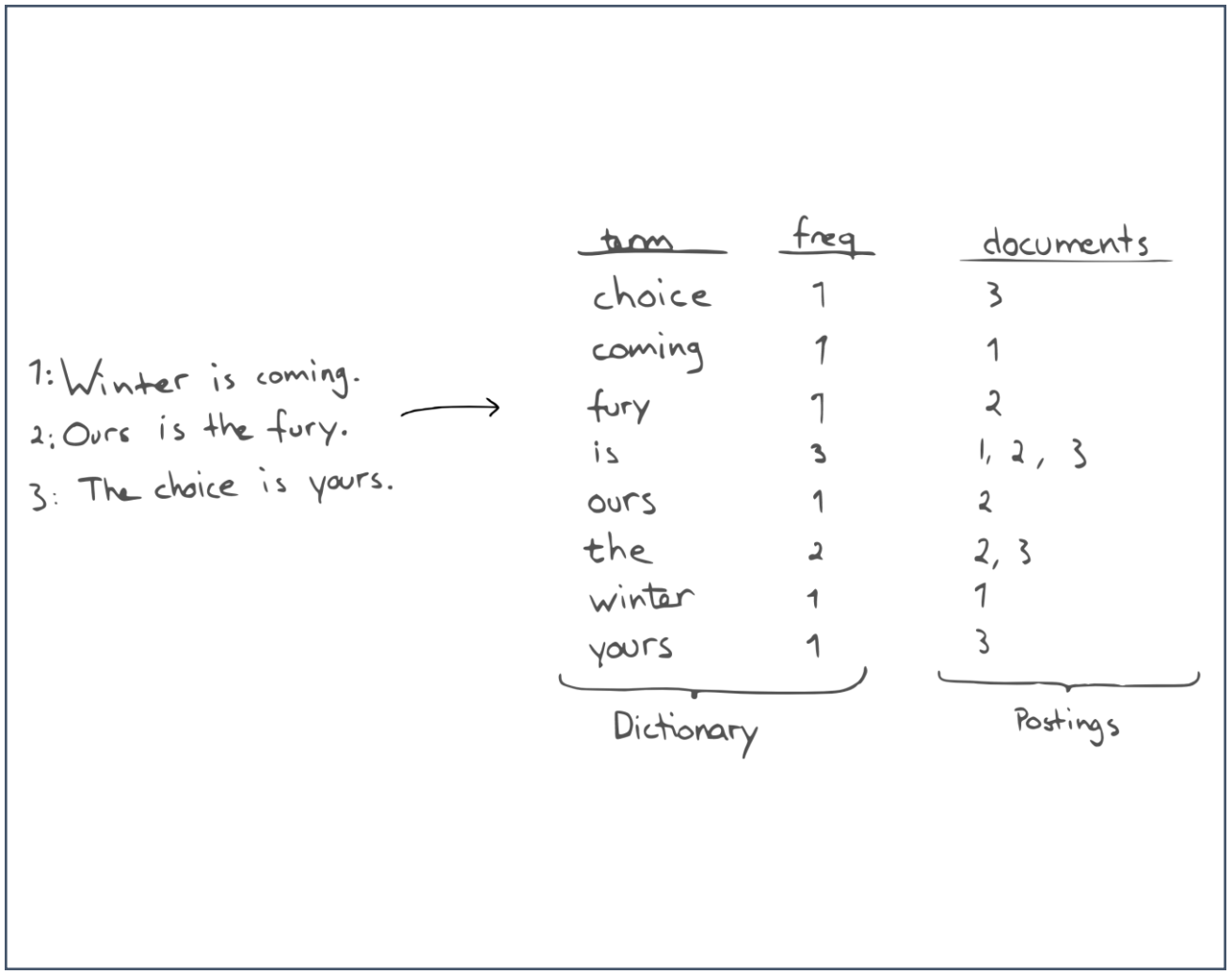
https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-create-index.html snippets/concepts/mappings.txt

# Invertierte Indizes

Ein invertierter Index ist die grundlegende Datenstruktur für Volltextsuche

- Dokumente werden in einzelne Terme in Kleinschreibung und ohne Zeichensetzung zerlegt
- Die Terme werden alphabetisch sortiert
- Die **freq** gibt an wie oft ein Term in allen Dokumenten vorkommt
- Die Spalte **documents** verweist auf die Dokumente in denen ein Begriff vorkommt, evtl. mit Offset im Dokument

Standardmäßig erstelle Elasticsearch einen invertierten Index für alle Felder eines Dokuments




https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up

# Elasticsearch

CRUD-Operationen

# CRUD-Operationen

Elasticsearch bietet CRUD-Operationen mit den folgenden APIs

- **C**reate – Index API
- **R**ead – Get API
- **U**pdate – Update API
- **D**elete – Delete API

Alle CRUD-Operationen beziehen sich auf einen einzelnen Index

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html

# Create

In Elasticsearch-Terminologie ist das Hinzufügen eines Dokuments zu einem Index eine Indexierungsoperation (engl. indexing operation)

- Für das Dokument wird der invertierte Index erstellt

Für ein Dokument kann mit PUT eine ID vorgegeben oder mit POST automatisch vergeben werden

```
PUT /books/_doc/1
{
  "title": "The Art of Computer Programming",
  "author": "Donald E. Knuth",
  "isbn": "978-0321751041",
  "pages": 3168,
  "price": 155.24
 }
```

```
{
  "_index": "books",
  "_type": "_doc",
  "_id": "1",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 0,
  "_primary_term": 1
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index_.html

snippets/crud/create.txt

# Read

Mit der Get API können Dokumente über ihre ID abgefragt werden

Die Get API funktioniert in Echtzeit und ist unabhängig von der Latenz der Indexierung von Dokumenten

Mit HEAD kann abgefragt werden ob ein Dokument mit der ID existiert

Über die `_source`-Route wird nur das `_source`-Feld geliefert

```
GET /books/_doc/1
{
  "_index": "books",
  "_type": "_doc",
  "_id": "1",
  "_version": 1,
  "_seq_no": 0,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "title": "The Art of Computer Programming",
    "author": "Donald E. Knuth",
    "isbn": "978-0321751041",
    "pages": 3168,
    "price": 155.24
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-get.html

snippets/crud/read.txt

# Update

Mit der Update API können Updates von Dokumenten geskripted und partielle Dokumente gemerged werden

Die Elasticsearch-Skriptsprache ist Painless

- https://www.elastic.co/guide/en/elasticsearch/painless/master/painless-guide.html

Aktualisierte Dokumente werden neu indiziert

Für komplette Updates sollte die Index API verwendet werden

Für Upsert-Operationen können mit `upsert` und `doc_as_upsert` initiale Dokumente spezifiziert werden

```
POST test/_update/1
{
  "script": {
    "source": "ctx._source.counter += params.count",
    "lang": "painless",
    "params": {
      "count": 4
    }
  }
}
```

```
{
  "_index": "test",
  "_type": "_doc",
  "_id": "1",
  "_version": 2,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 1,
  "_primary_term": 1
}
```

```
POST test/_update/1
{
  "doc": {
    "name": "new_name"
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-update.html

snippets/crud/update.txt

# Delete

Mit der Delete API können Dokumente über ihre ID entfernt werden

Mit `version` und `version_type` können nebenläufige Zugriffe gesteuert werden

```
DELETE /test/_doc/2?version=3&version_type=external
```

```
{
  "_index": "test",
  "_type": "_doc",
  "_id": "2",
  "_version": 3,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 11,
  "_primary_term": 1
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-delete.html

snippets/crud/delete.txt

# Nebenläufigkeit

Das Feld `_primary_term` erhöht sich jedes Mal wenn ein Shard primary wird.

→ Löse Änderungen auf wenn ein alter Primary Shard zurück kommt

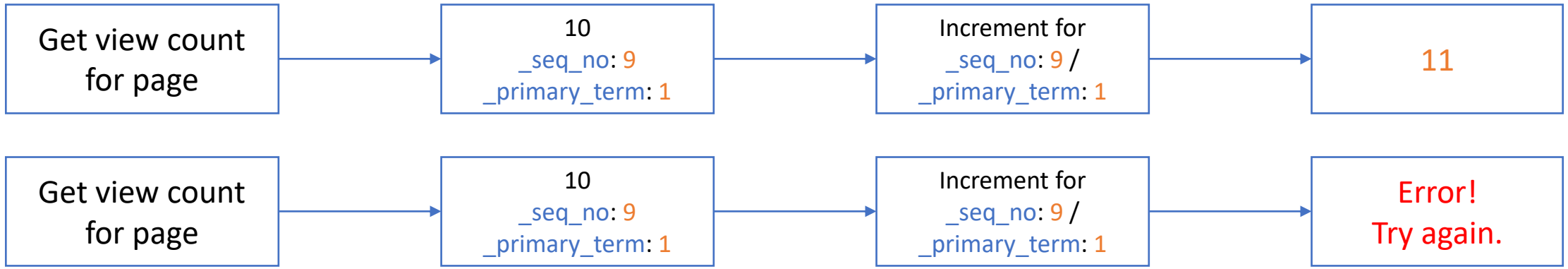
Das Feld `_seq_no` erhöht sich jedes Mal wenn in dem Index geschrieben wird.

```
GET movies/_doc/1

PUT movies/_doc/1
{
  "title": "Toy Story"
}
```

```
PUT movies/_doc/1?if_seq_no=9117&if_primary_term=3
{
  "title": "Toy Story New"
}

POST movies/_update/1?retry_on_conflict=5
{
  "doc":
  {
    "title": "Toy Story Newest“
  }
}
```




Use `retry_on_conflicts = N` to automatically retry.

https://www.elastic.co/guide/en/elasticsearch/reference/current/optimistic-concurrency-control.html

# Elasticsearch

Indexe & Abbildungen

# Erzeugung eines Index

Elasticsearch erzeugt Indizes und die Abbildung auf die Datentypen automatisch

→ Dynamic Mapping

```
PUT /books/_doc/1
{
  "title": "The Art of Computer Programming",
  "genre": "science"
}

GET /books/_mapping
```

```
{
  "books": {
    "mappings": {
      "properties": {
        "genre": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
  ...
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-create-index.html snippets/indexes/automatic.txt

# Erzeugung eines Index

Aber: Indizes und deren Abbildungen können auch definiert werden

→ Explicit Mapping

```
PUT /books
{
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "properties": {
      "title": { "type": "text" },
      "genre": { "type": "keyword",
      "ignore_above": 100 }
    }
  }
}

GET /books/_mapping
```

```
{
  "books": {
    "mappings": {
      "properties": {
        "genre": {
          "type": "keyword",
          "ignore_above": 100
        },
        "title": {
          "type": "text"
        }
      ...
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-create-index.html

snippets/indexes/define.txt

# Abbildungen & Multi-Felder

Die Abbildungen (engl. mappings) definieren, wie Dokumente und deren Felder gespeichert und indiziert werden

Mit der **Put Mapping API** können einem Index Abbildungen hinzugefügt und Abbildungen geändert werden

- https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-put-mapping.html

Multi-Felder (engl. multi-fields) werden mehrfach indiziert

Außer bei Änderungen ausgewählter Abbildungsparameter erfordern Änderungen von Abbildungen eine neue Indizierung

```
PUT /books/_doc/1
{
  "title": "The Art of Computer Programming",
  "genre": "science"
}

GET /books/_mapping
```

```
{
  "books": {
    "mappings": {
      "properties": {
        "genre": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
  ...
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

snippets/indexes/mappings.txt
snippets/indexes/reindex.txt

# Suche in mehreren Indizes

Die folgende Suchanfrage findet alle Dokumente in allen Indizes

- `GET /_search`

Mit der Option size kann die Anzahl der Suchtreffer begrenzt werden

- `GET /_search?size=100`

Für eine Suche in mehreren Indizes werden diese in der Request URL komma-separiert aufgelistet

```
PUT /persons/_doc/1
{
  "name": "Donald E. Knuth"
}

PUT /authors/_doc/1
{
  "name": "Donald Ervin Knuth"
}

GET /persons,authors/_search
```

```
"hits": [
    {
      "_index": "authors",
      "_type": "_doc",
      "_id": "1",
      "_score": 1.0,
      "_source": {
        "name": "Donald Ervin Knuth"
      }
    },
    {
      "_index": "persons",
      "_type": "_doc",
      "_id": "1",
      "_score": 1.0,
      "_source": {
        "name": "Donald E. Knuth"
      }
    }
]
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

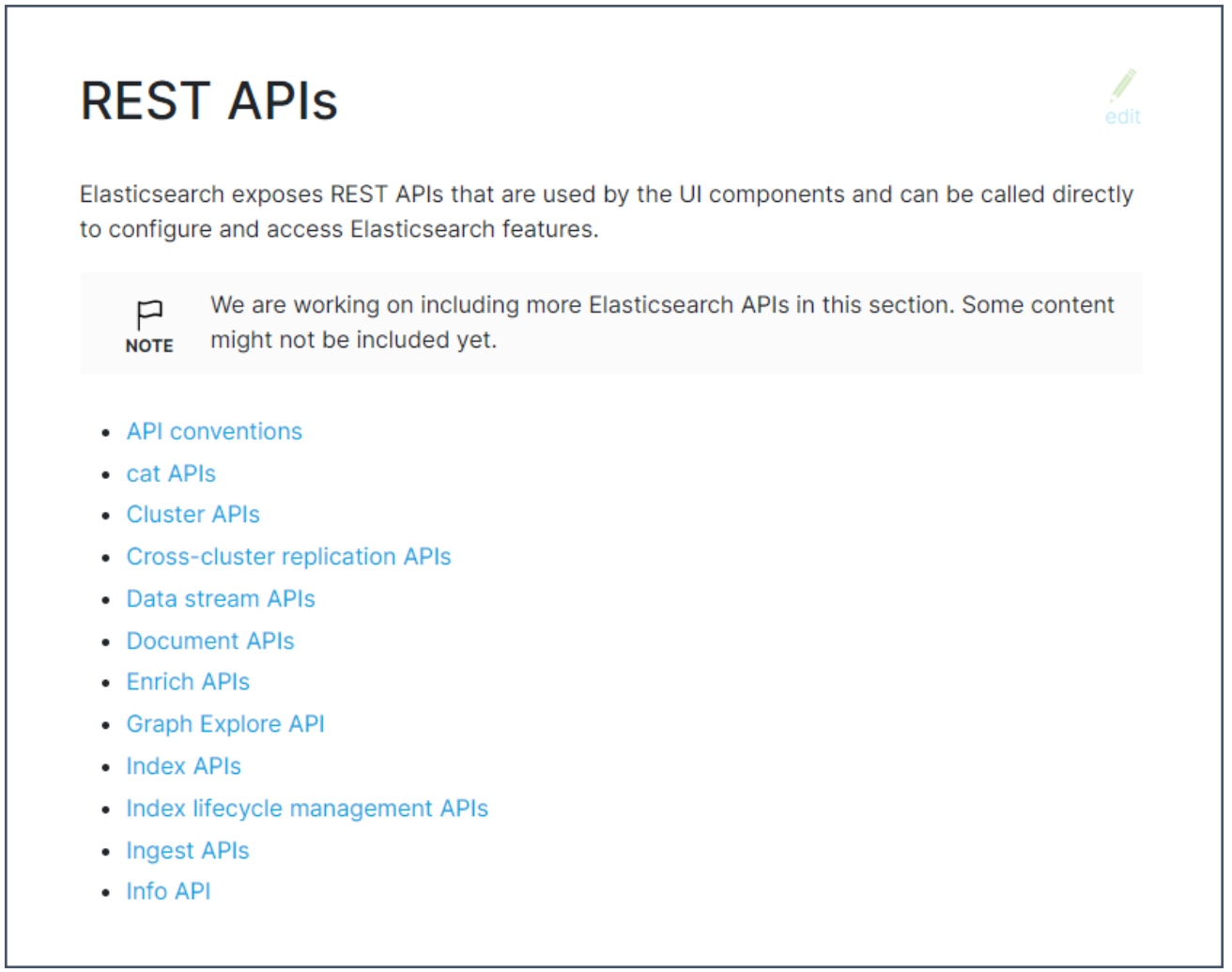
snippets/indexes/multiple.txt

# Elasticsearch REST APIs

Elasticsearch bietet mehr als 20 REST APIs

Unter anderem die…

- Document APIs für CRUD-Operationen
- Index APIs für das Erstellen und Bearbeiten von Indizes
- Search API für Suche und Aggregation
- Cluster APIs für Clustermanagement
- cat API für einfache Interaktion mit Elasticsearch https://www.elastic.co/blog/introducing-cat-api



REST APIs

edit

Elasticsearch exposes REST APIs that are used by the UI components and can be called directly to configure and access Elasticsearch features.

NOTE

We are working on including more Elasticsearch APIs in this section. Some content might not be included yet.

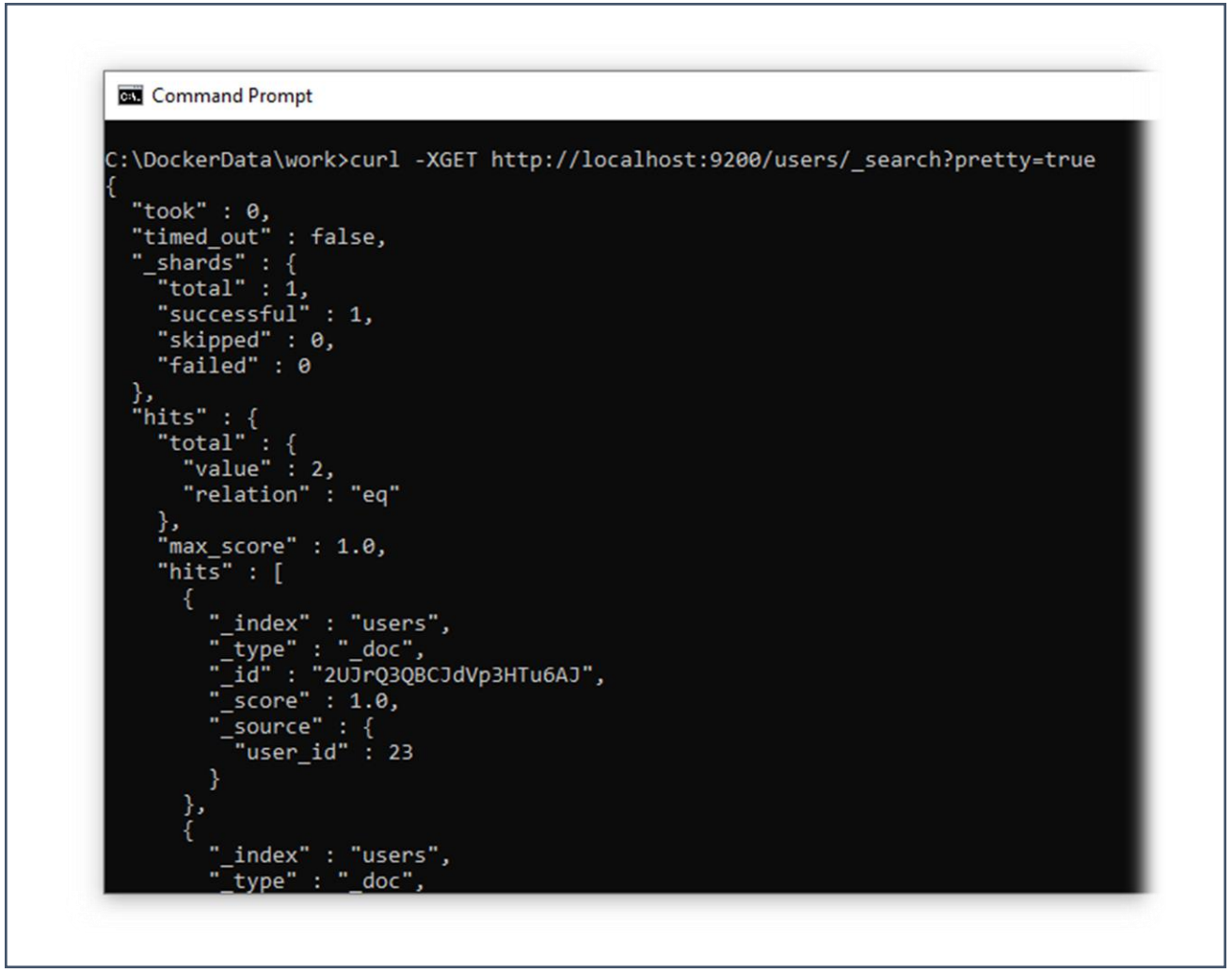
- API conventions
- cat APIs
- Cluster APIs
- Cross-cluster replication APIs
- Data stream APIs
- Document APIs
- Enrich APIs
- Graph Explore API
- Index APIs
- Index lifecycle management APIs
- Ingest APIs
- Info API

https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html

# Formatierung der JSON-Antworten

Mit der Option `pretty=true` werden JSON-Antworten formatiert

Die Darstellung in den Kibana Dev Tools ist automatisch formatiert

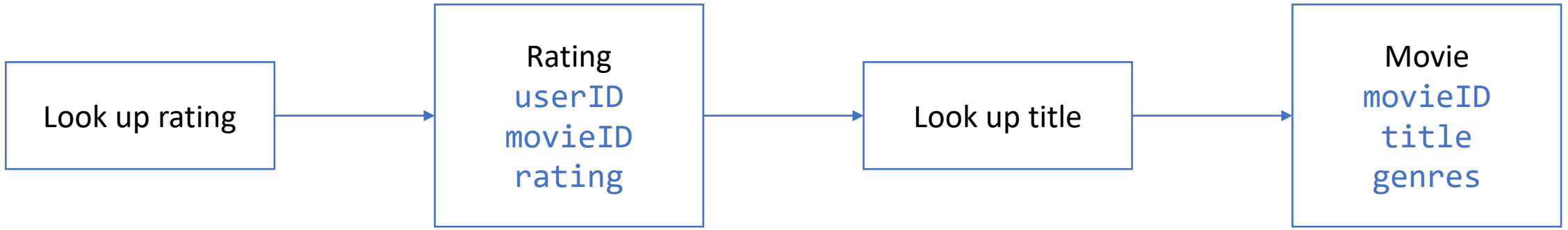


# Datenmodellierung

Normalisierung minimiert den Speicherbedarf…

… aber es sind zwei Abfragen erforderlich

… und Speicher ist günstig



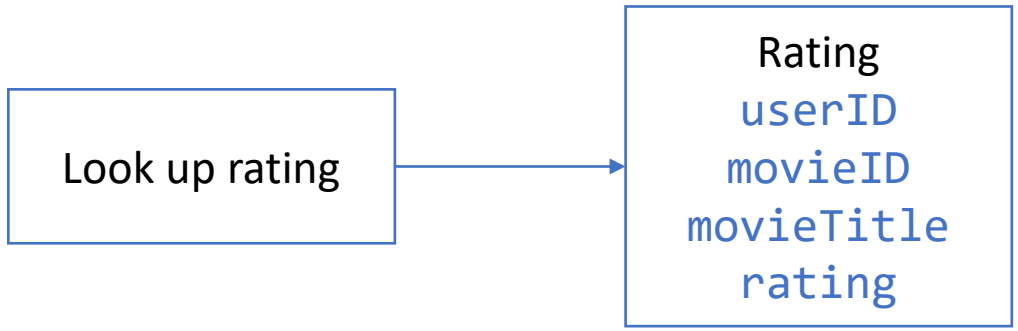
# Datenmodellierung

Denormalisierte Datenspeicherung…

… erfordert mehr Speicher für die duplizierten Filmtitel

… aber erfordern nur eine Abfrage

(die Filmtitel sollten konstant sein)



# Flattened Field Type

Zum Beispiel bei Log-Einträgen können viele unterschiedlich strukturierte JSON-Objekte anfallen

Elasticsearch erzeugt jeweils dynamisch ein Mapping in dem Cluster-Zustand

Jedes neue Feld…

… vergrößert den Cluster-Zustand

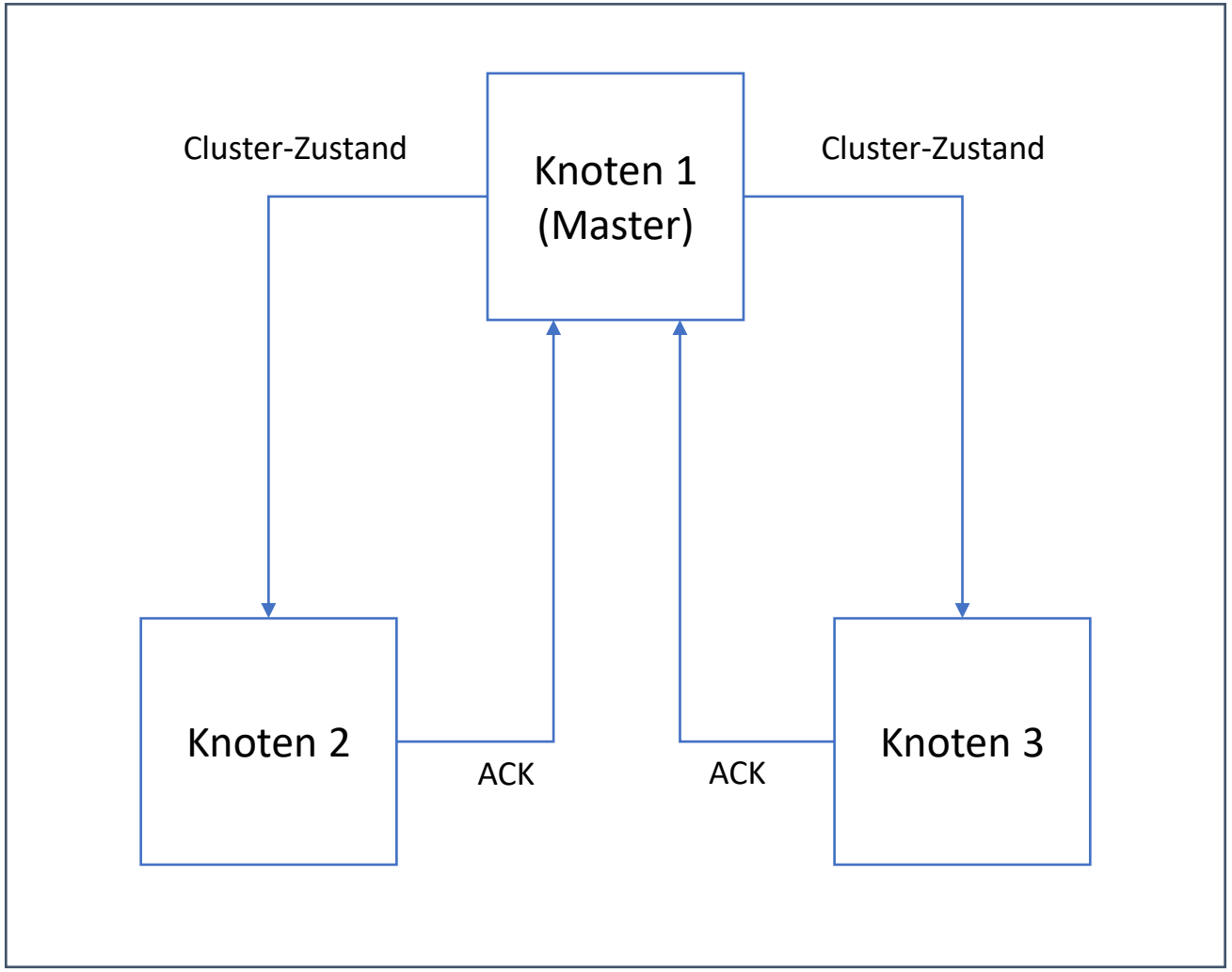
… und der Cluster-Zustand muss von dem Master-Knoten synchronisiert werden

Ohne synchronisierten Cluster-Zustand können die Knoten zum Beispiel keine Indexing- oder Such-Operationen ausführen

→ Mapping Explosion

Bei einem Flattened-Datentyp erscheinen die geschachtelten Felder nicht in dem Mapping

Die Felder eines Flattened-Datentyps werden als `keyword`-Felder behandelt (keine Analyzer,…)!




https://www.elastic.co/guide/en/elasticsearch/reference/current/flattened.html

snippets/flattened

# Mapping Exceptions

Werte die nicht zu einem definierten Feld-Datentyp passen führen zu einer…

Mapping Exception

( `mapper_parsing_exception` )

Es findet eine Typumwandlung statt

"5" == 5

Ein Workaround ist die nicht-dynamische Index-Eigenschaft

`index.mapping.ignore_malformed`

Nicht ignoriert werden JSON-Objekte!

Achtung: Standardmäßig ist ein Mapping auf 1000 Felder beschränkt

```
PUT microservice-logs
{
   "mappings": {
       "properties": {
           "timestamp": { "type": "date"  },
           "service": { "type": "keyword" },
           "host_ip": { "type": "ip" },
           "port": { "type": "integer" },
           "message": { "type": "text" }
       }
   }
}
```

```
POST microservice-logs/_doc?pretty
{
  "timestamp": "2020-04-11T12:34:56.789Z",
  "service": "XYZ",
  "host_ip": "10.0.2.15",
  "port": "NONE",
  "message": "I am not well!"
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-ignored-field.html

snippets/exceptions

# Suche – Was ist relevant?

Grundlagen

# Grundlagen der Suche in Text

Suche in Zahlen oder Datumsangaben ist definitiv

Text kann strukturiert und unstrukturiert sein

Strukturierter Text können kategorische Werte sein wie zum Beispiel…

- Ländercodes
- Produktklassen

Auch bei kategorischen Werten ist oft eine perfekte Übereinstimmung mit einem Suchbegriff gefragt

Bei unstrukturiertem Text ist das Suchergebnis eine nach Relevanz sortierte Liste der gefundenen Dokumente

# Textanalyse

Textanalyse (engl. text analysis) ist die Umwandlung von unstrukturiertem Text in eine für die Suche optimierte Form

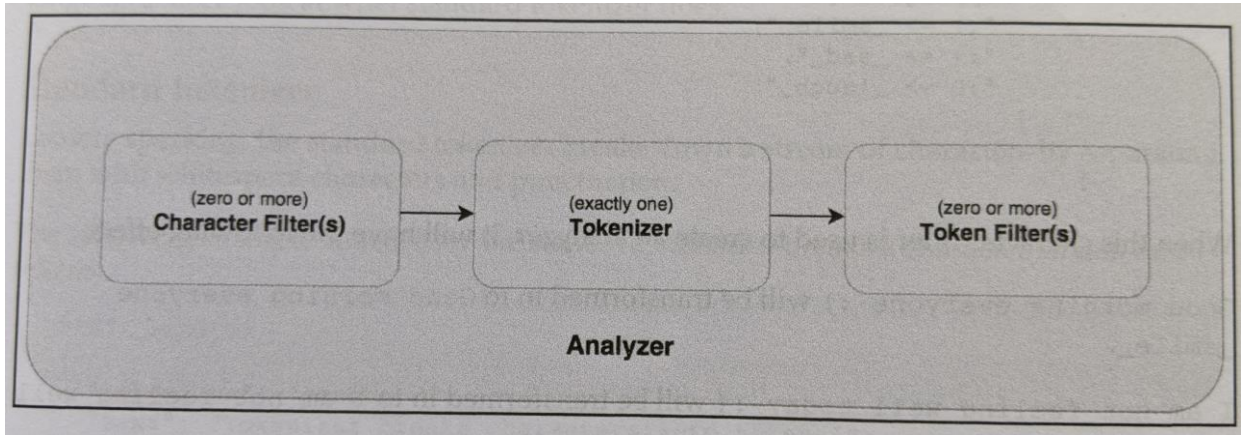
Elasticsearch analysiert Felder vom Typ `text`

Beispiel

- Suche in „The quick brown fox jumps over the lazy dog“
- „Quick fox jumps“
- „foxes leap“

Erforderlich sind Tokenisierung (engl. tokenization) und Normalisierung (engl. normalization)

- Quick → quick (Kleinschreibung)
- foxes → fox (Stemming)
- leap → jump (Synonyme)




https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html

# Elasticsearch Analyzers

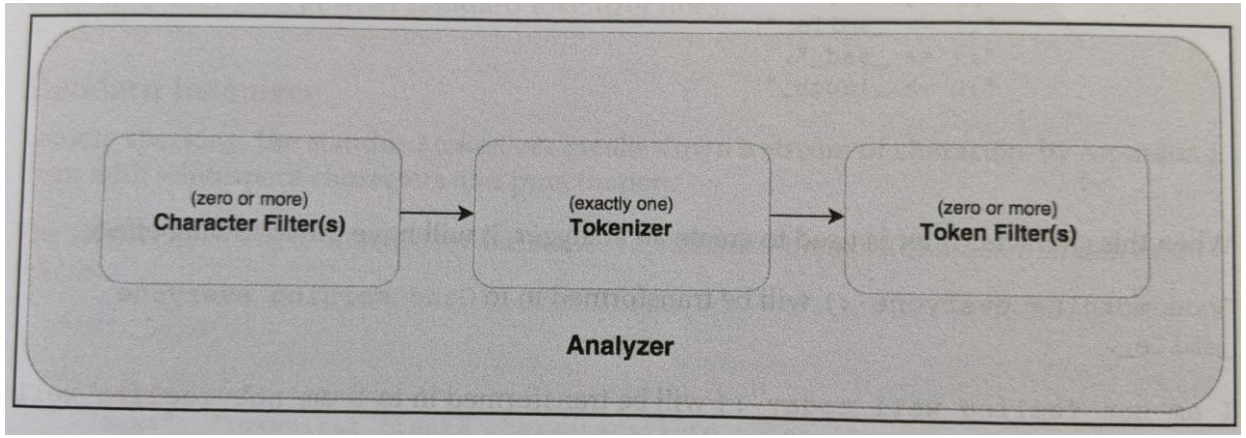
Ein Elasticsearch Analyzer zerlegt Texte in einzelne, normalisierte Terme (Tokens)

- Bei der Indexierung
- Bei der Suche

Elasticsearch Analyzers werden pro `text`-Feld konfiguriert

Ein Elasticsearch Analyzer besteht aus den folgenden Komponenten

- Character Filter (0 - *)
- Tokenizer (1)
- Token Filter (0 - *)




https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html

# Elasticsearch Analyzers

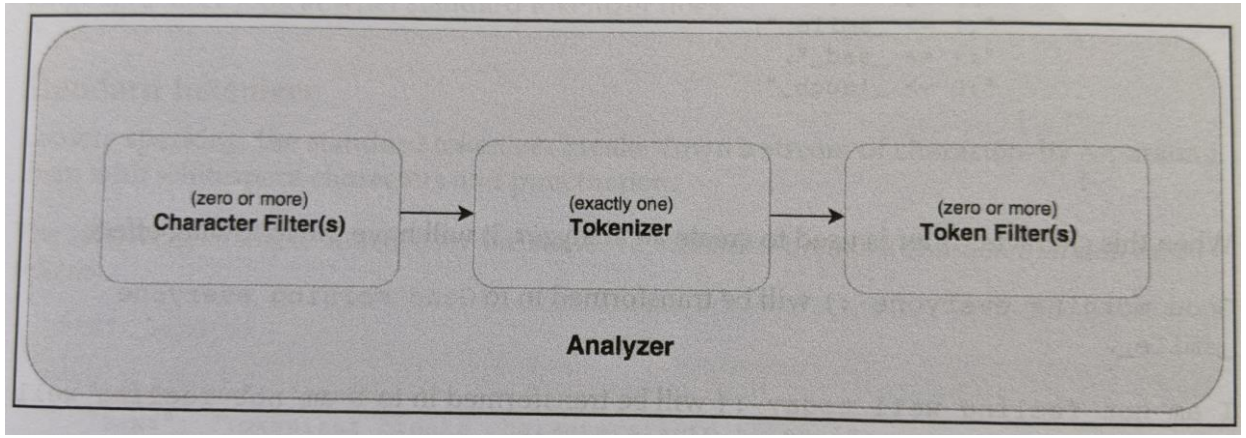
Standardmäßig verwendet Elasticsearch einen Standard Analyzer

Weitere Built-in Analyzers stehen zur Verfügung

Auch eigene Analyzers können konfiguriert werden

Ein Analyzer steuert die folgenden Schritte

- Änderungen an dem Text vor der Tokenisierung
- Tokenisierung
- Normalisierung und Filterung der Tokens




https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html

# Analyze API

Die Analyze API analysiert einen gegebenen Text und gibt die produzierten Tokens zurück

```
GET /_analyze
{
  "analyzer": "standard",
  "text": "Quick Brown Foxes!"
}
```

```
{
  "tokens": [
    {
      "token": "quick",
      "start_offset": 0,
      "end_offset": 5,
      "type": "<ALPHANUM>",
      "position": 0
    },
    ...
  ]
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-analyze.html

snippets/analysis/analyze.txt

# Character Filters

Ein Character Filter kann in einem Zeichenstrom (engl. character stream) Zeichen löschen, ersetzen und hinzufügen

Elasticsearch enthält verschiedene eingebaute Character Filters

- Mapping Character Filter
- Pattern Replace Character Filter
- HTML Strip Character Filter

Übung: Ersetze mit einem Character Filter wie folgt.

- :) → _smile_
- :( → _sad_
- :D → _laugh_

```
GET /_analyze
{
  "tokenizer": "keyword",
  "char_filter": ["html_strip"],
  "text": "<p>I&apos;m so <b>happy</b>!</p>"
}
```

```
{
  "tokens" : [
    {
      "token" : """
I'm so happy!
""",
      "start_offset" : 0,
      "end_offset" : 32,
      "type" : "word",
      "position" : 0
    }
  ]
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-charfilters.html snippets/analysis/character_filter.txt

# Tokenizer

Ein Elasticsearch Analyzer hat genau einen Tokenizer

Ein Tokenizer erzeugt aus einem Zeichenstrom einen Strom von Tokens

Ein Token kann zum Beispiel ein Wort aus dem Dokument sein

Aus Tokens wird der invertierte Index erstellt

Ein Token enthält…

- Die Position des Tokens im Tokenstrom
- Start und Ende der entsprechenden Zeichen im Eingabedokument
- Den Typ des Tokens (z.B. WORD)

```
GET /_analyze
{
  "tokenizer": "standard",
  "text": "The 2 QUICK Brown-Foxes
           jumped over the lazy dog's bone."
}
```

```
{
  "tokens" : [
    {
      "token" : "The",
      "start_offset" : 0,
      "end_offset" : 3,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    ...
  ]
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html

snippets/analysis/tokenizer.txt

# Wort-basierte Tokenizer

Die folgenden Tokenizer zerlegen einen Text in einzelne Wörter

- Standard Tokenizer
- Letter Tokenizer
- Lowercase Tokenizer
- Whitespace Tokenizer
- UAX URL Email Tokenizer
- Classic Tokenizer
- Thai Tokenizer

Einige Tokenizer verwenden den „Unicode Text Segmentation“-Algorithmus

- https://unicode.org/reports/tr29/#Grapheme_Cluster_Boundaries

```
GET /_analyze
{
  "tokenizer": "standard",
  "text": "The 2 QUICK Brown-Foxes
           jumped over the lazy dog's bone."
}
```

```
{
  "tokens" : [
    {
      "token" : "The",
      "start_offset" : 0,
      "end_offset" : 3,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    ...
  ]
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html

snippets/analysis/tokenizer.txt

# Token-Filter

Ein Token-Filter verarbeitet einen Strom von Tokens und kann…

- Tokens modifizieren (z.B. Lowercasing)
- Tokens entfernen (z.B. Stopwörter)
- Tokens hinzufügen (z.B. Synonyme)

Elasticsearch enthält eine Reihe eingebauter Token-Filter wie z.B.

- Classic
- Unique
- Linguistische/semantische Filter
  - Hunspell, Normalization, Synonym

```
GET /_analyze
{
  "tokenizer": "classic",
  "filter": ["classic"],
  "text": "The 2 Q.U.I.C.K. Brown-Foxes jumped
           over the lazy dog's bone."
}
```

```
{
  "tokens": [
    ...
    {
      "token": "QUICK",
      "start_offset": 6,
      "end_offset": 16,
      "type": "<ACRONYM>",
      "position": 2
    }
    ...
  ]
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenfilters.html snippets/analysis/token_filter.txt

# Elasticsearch Analyzers

Elasticsearch enthält eine Reihe eingebauter Analyzer wie z.B.

- Standard Analyzer
- Simple Analyzer
- Whitespace Analyzer
- Language Analyzers

Die Analyzer können mit verschiedenen Parametern konfiguriert werden

Analyzer werden pro Feld konfiguriert

Für `text`-Felder verwendet Elasticsearch standardmäßig den Standard Analyzer

```
PUT my-index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_english_analyzer": {
          "type": "standard",
          "stopwords": "_english_"
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "my_text": {
        "type": "text",
        "analyzer": "my_english_analyzer"
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html

# Custom Analyzers

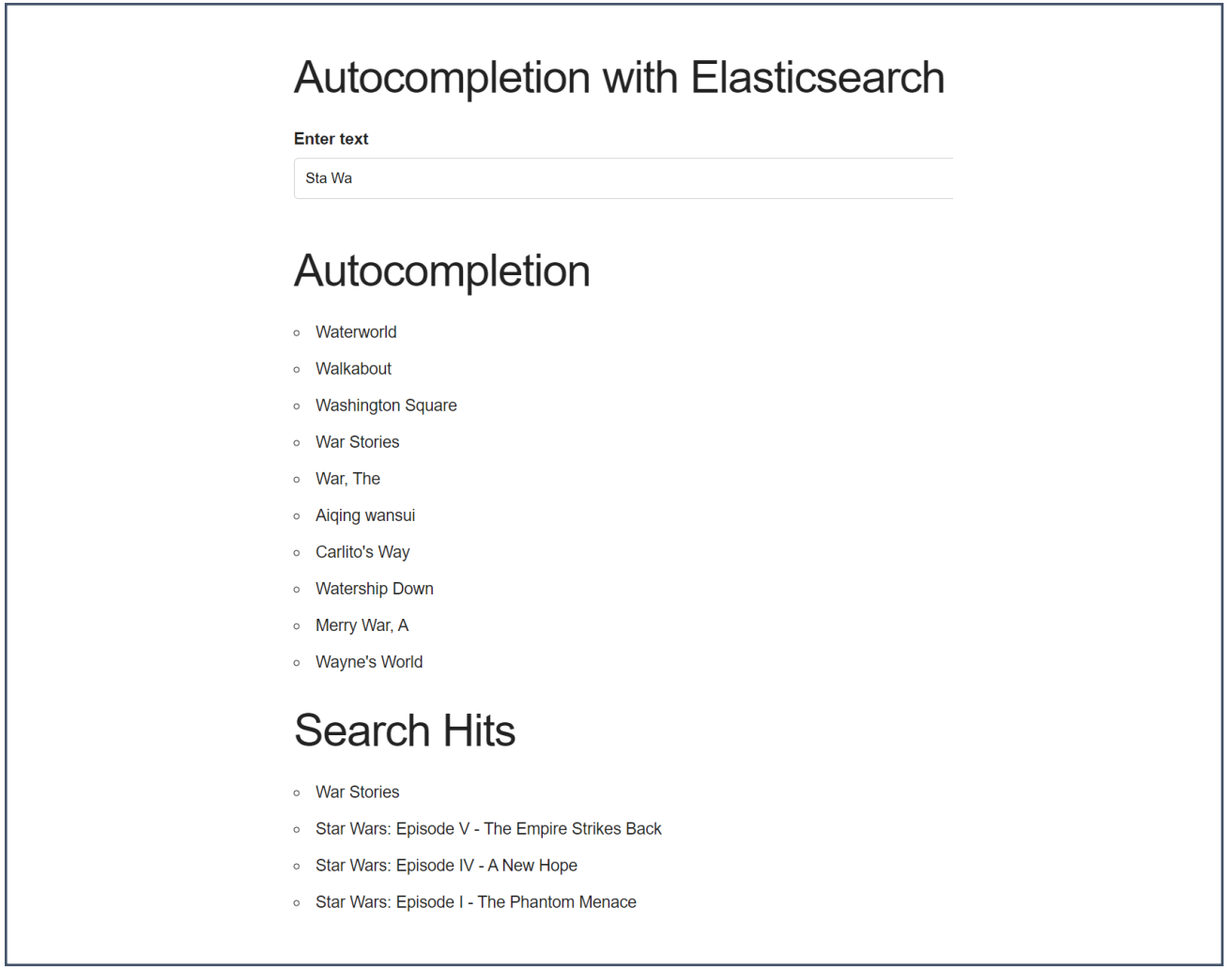
In manchen Fällen sind eigene Kombinationen von Character Filter, Tokenizer und Token Filter erforderlich

Beispiel: Autocompletion und die Suche mit unvollständigen Eingaben

- Suche nach „Sta“
- Suche nach „Sta Wa“

Unvollständige Eingaben sind N-Gramme

- N-Gramme sind das Ergebnis der Zerlegung eines Textes in Fragmente.
- Der Text wird dabei zerlegt, und jeweils *N* aufeinanderfolgende Fragmente werden als *N*-Gramm zusammengefasst.




https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-edgengram-tokenfilter.html

apps/autocompletion/

# Suche – Was ist relevant?

Suche in strukturierten Daten

# Suche in strukturierten Daten

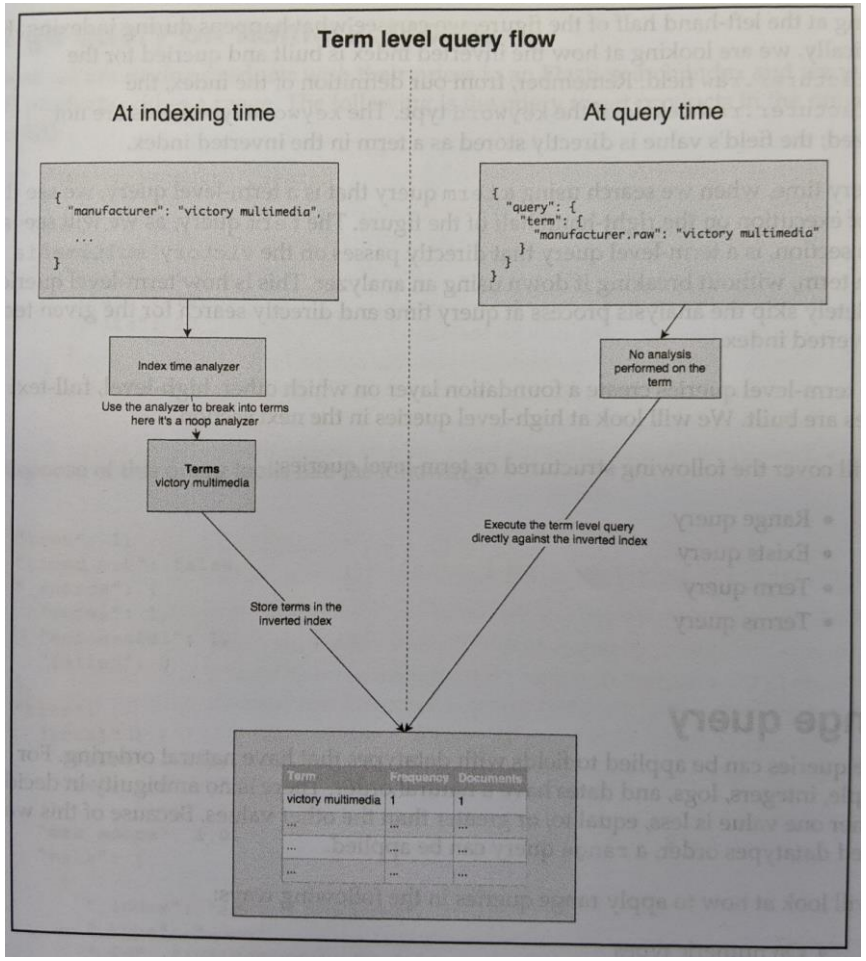
Bisher: Relevanz-basierte Suche mit einer Metrik und einer Sortierung, wie gut ein Dokument zu einer Anfrage passt

Bei strukturierten Daten binäre Entscheidung ob ein Dokument in der Ergebnismenge enthalten sein soll oder nicht

Strukturierte Suchanfragen (engl. structured search queries) werden auch als term-level queries bezeichnet

Verschiedene term-level-queries

- Range
- Exists
- Term(s)



# Range Queries

Eine Range Query liefert Dokumente mit Eigenschaften innerhalb eines Wertebereichs

Einige Attribute des Ergebnis-Dokuments

- `hits.total.value`
  Die Anzahl der Suchtreffer
- `hits.max_score`
  Der Score des besten Suchtreffers, bei strukturierten Suchanfragen immer 1
- `hits.hits`
  Ein Array mit den Ergebnisdokumenten

Range Queries können für Datentypen mit Wertebereichen und partieller Ordnung definiert werden

```
GET /amazon/_search
{
  "query": {
    "range": {
      "price": {
        "gte": 10,
        "lte": 20
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-range-query.html

snippets/structured-data/range_queries.txt

# Score Boosting

Die Suchtreffer einer Range Query haben immer den Score 1

Für zusammengesetzte Suchanfragen (engl. compound queries) können angepasste Scores sinnvoll sein

Mit dem `boost`-Parameter kann bei Range Queries der Score angepasst werden

```
GET /amazon/_search
{
  "query": {
    "range": {
      "price": {
        "gte": 10,
        "lte": 20,
        "boost": 2.5
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-range-query.html

snippets/structured-data/range_queries.txt

# Datumsangaben

JSON kennt keine Datumsangaben

Verschiedene Darstellungen in Elasticsearch

- Strings mit formatierten Datumsangaben wie "2015/01/01 12:10:30"
- Long-Werte für die milliseconds-since-the-epoch
- Integer-Werte für die seconds-since-the-epoch

Aktueller Zeitpunkt `now`

Abkürzungen für Datumskomponenten

Addition und Subtraktion von Zeit, z.B. `now - 1y` ist „heute vor einem Jahr“

Rundung z.B. auf Tage mit `/d` (`now - 7d/d`)

```
GET /artists/_search
{
  "query": {
    "range": {
      "birthday": {
        "gte": "01/01/1940",
        "lte": "12/31/1959",
        "format": "MM/dd/yyy"
      }
    }
  }
}
```

Datumskomponenten:

```
y (year), M (month), d (day),
h / H (hour), m (minutes), s (seconds)
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/date.html

snippets/structured data/range_queries.txt

# Exists Queries

Eine Exists Query liefert Dokumente mit einem indexierten Wert für ein Feld

Ein indexierter Wert ist nicht vorhanden wenn…

- Das Feld Eingabe-JSON ist `null` oder `[]`
- Das Feld ist nicht indexiert
- Das Feld hat einen zu langen oder ungültigen Wert

Exists Queries werden in einem Filter-Kontext (engl. filter context) ausgeführt

- Es findet kein Scoring der Suchtreffer statt
- Jeder Suchtreffer erhält den Score 1
- Caching der Suchtreffer als Array (0/1)

```
GET /artists/_search
{
  "query": {
    "exists": {
      "field": "birthday"
    }
  }
}
```

```
GET /artists/_search
{
  "query": {
    "bool": {
      "must_not": {
        "exists": {
          "field": "birthday"
        }
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-exists-query.html

snippets/structured-data/exist_queries.txt

# Term Queries

Eine Term Query liefert Dokumente mit einem *exakten* Term für ein Feld

- Es findet keine Analyse statt

Achtung: Term Queries sollten nicht mit `text`-Feldern verwendet werden

- Analyzers verändern die Werten von `text`-Feldern
- Suche in `text`-Feldern mit `match`

Term Queries werden standardmäßig im Query Context ausgeführt

- Elasticsearch berechnet Scores

Beschleunigte Abfragen mit einer Ausführung als `constant_score`-Abfrage

```
GET /amazon/_search
{
  "query": {
    "term": {
      "manufacturer.raw": "victory multimedia"
    }
  }
}
```

```
GET /amazon/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "term": {
          "manufacturer.raw": "victory multimedia"
        }
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-term-query.html

snippets/structured data/term_queries.txt

# Suche – Was ist relevant?

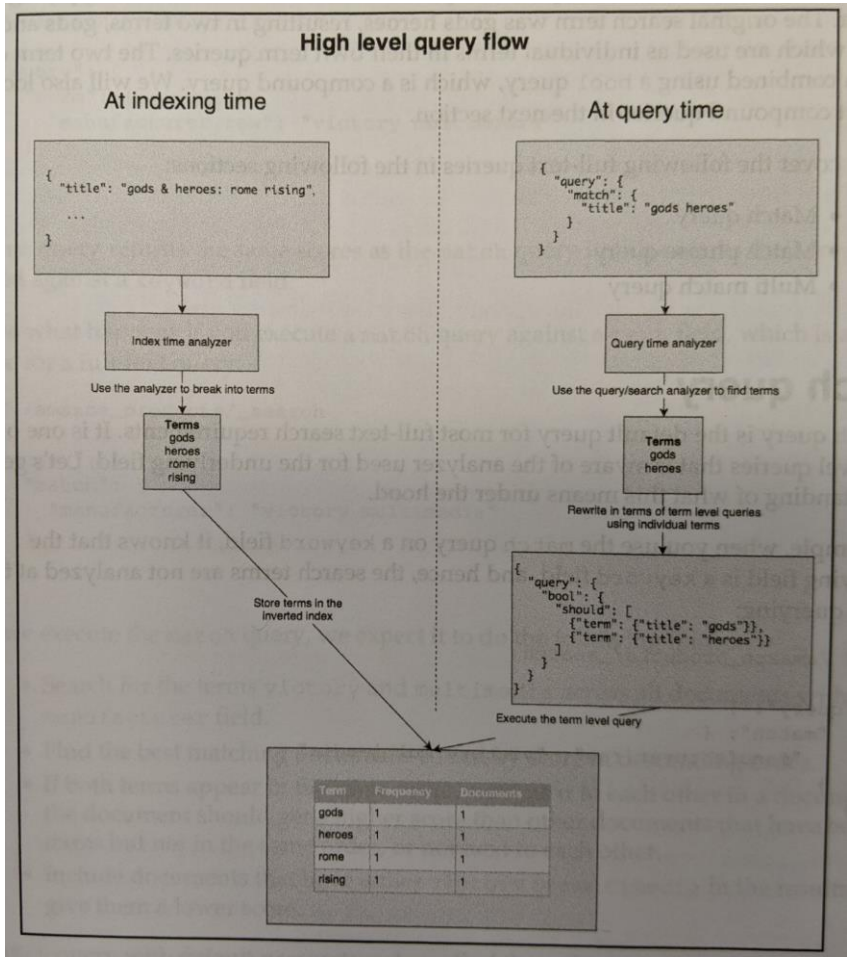
Volltextsuche

# Volltextsuche

Anfragen für eine Volltextsuche basieren auf dem Analyseprozess mit Character Filter, Tokenizer und Token Filter

Anfragen basierend auf dem Analyseprozess sind High-level Queries

Analyzer werden für die Indexierung *und* die Suche angewendet



# Search Analyzer

Normalerweise wird der gleiche Analyzer für die Indexierung und die Suche verwendet

- Indexierte Terme und Suchterme müssen im gleichen Format sein

```
PUT /my-index/_doc/1
{
  "text": "Quick Brown Fox"
}

GET /my-index/_search
{
  "query": {
    "match": {
      "text": {
        "query": "Quick Br",
        "operator": "and"
      }
    }
  }
}
```

```
PUT /my-index
{
  "settings": {
    "analysis": {
      "filter": {
        "autocomplete_filter": {
          "type": "edge_ngram",
          "min_gram": 1,
          "max_gram": 20
        }
      },
      "analyzer": {
        "autocomplete": {
          "type": "custom",
          "tokenizer": "standard",
          "filter": ["lowercase", "autocomplete_filter"]
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "text": {
        "type": "text",
        "analyzer": "autocomplete",
        "search_analyzer": "standard"
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-analyzer.html

snippets/full-text search/search_analyzer.txt

# Match Queries

Match Queries sind High-level Queries für Volltextsuchen

Match Queries basieren auf dem Analyseprozess, außer bei `keyword`-Feldern

- Umwandlung in eine Term Query mit den gleichen Scores

Standardmäßiges Verhalten von Match Queries auf `text`-Feldern:

- Suche der einzelnen Terme in allen Dokumenten mit `manufacturer`-Feld
- Sortierung der Suchtreffer nach Relevanz
- Suchtreffer mit den Suchtermen in der angegebenen Reihenfolge sind relevanter
- Suchtreffer mit nur einzelnen Suchtermen sind weniger relevant

```
GET /amazon/_search
{
  "query": {
    "match": {
      "manufacturer.keyword": "victory multimedia"
    }
  }
}

GET /amazon/_search
{
  "query": {
    "term": {
      "manufacturer.keyword": "victory multimedia"
    }
  }
}

GET /amazon/_search
{
  "query": {
    "match": {
      "manufacturer": "victory multimedia"
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html

snippets/full-text search/match_queries.txt

# Match Queries - Optionen

`operator`

Logische Verknüpfung der Suchbegriffe mit AND oder OR

`minimum_should_match`

Minimale Anzahl an Suchbegriffen, die in Suchtreffern enthalten sein müssen

`fuzziness`

Maximale Editierdistanz von Suchtreffern

- Mögliche Werte 0, 1, 2, AUTO
- `max_expansions` definiert die maximale Anzahl an Varianten
- `prefix_length` definiert einen Präfix von dem keine Varianten gebildet werden

```
GET /amazon/_search
{
  "query": {
    "match": {
      "manufacturer": {
        "query": "victory multimedia",
        "operator": "and"
      }
    }
  }
}

GET /amazon/_search
{
  "query": {
    "match": {
      "manufacturer": {
        "query": "victory multimedia",
        "minimum_should_match": 2
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#fuzziness

# Levenshtein-Distanz

Die Levenshtein-Distanz (auch Editierdistanz) zwischen zwei Zeichenketten ist die minimale Anzahl von Einfüge-, Lösch- und Ersetz-Operationen, um die erste Zeichenkette in die zweite umzuwandeln

Berechnung mit der Methode der dynamischen Programmierung

- Teile das Problem in Teilprobleme
- Speichere und verwende die Zwischenresultate

Vielfältige Anwendungen

- Phonetische Suche durch Anwendung auf Lautsymbol-Zeichenketten
- Schreibmaschinendistanz für Korrekturvorschläge

$$
\begin{aligned}
&m = |u| \\
&n = |v| \\
&D_{0,0} = 0 \\
&D_{i,0} = i, 1 \le i \le m \\
&D_{0,j} = j, 1 \le j \le n \\
&D_{i,j} = \min \begin{cases} D_{i-1,j-1} & +0 \text{ falls } u_i = v_j \\ D_{i-1,j-1} & +1 \text{ (Ersetzung)} \\ D_{i,j-1} & +1 \text{ (Einfügung)} \\ D_{i-1,j} & +1 \text{ (Löschung)} \end{cases}, \\
&\quad 1 \le i \le m, 1 \le j \le n
\end{aligned}
$$

| | ε | T | o | r |
|---|---|---|---|---|
| ε | 0 | 1 | 2 | 3 |
| T | 1 | 0 | 1 | 2 |
| i | 2 | 1 | 1 | 2 |
| e | 3 | 2 | 2 | 2 |
| r | 4 | 3 | 3 | 2 |

https://de.wikipedia.org/wiki/Levenshtein-Distanz

# Match Phrase Queries

Match Phrase Queries berücksichtigen die Reihenfolge der Suchbegriffe innerhalb eines Ausdrucks

Der Parameter `slop` spezifiziert wie viele Begriffe in den Suchanfragen fehlen dürfen (d.h. in den Suchtreffern zusätzlich auftreten dürfen)

```
GET /amazon/_search
{
  "query": {
    "match_phrase": {
      "description": {
        "query": "real saltware aquarium on your desktop",
        "slop": 1
      }
    }
  }
}

GET /amazon/_search
{
  "query": {
    "match": {
      "description": {
        "query": "real saltware aquarium on your desktop"
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query-phrase.html

snippets/full-text search/match_phrase_queries.txt

# Multi Match Queries

Für Multi Match Queries sucht Elasticsearch in einer Menge spezifizierter Felder

- Suchtreffer in bestimmten Feldern können mit per field boosting höher gewichtet werden
- Feldnamen können mit Wildcards (‚*‘) spezifiziert werden
- Sind keine Felder spezifiziert sucht Elasticsearch im `index.query.default_field`
  - Standardmäßig ist das *.
  - * extrahiert alle Felder eines Index auf die `term`-queries angewendet werden können.

```
GET /amazon/_search
{
  "query": {
    "multi_match": {
      "query": "monitor aquarium on your desktop",
      "fields": ["title^3", "description"]
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-match-query.html

snippets/full-text search/multi_match_queries.txt

# Suche – Was ist relevant?

Compound Queries

# Compound Queries

Compound Queries…

- kombinieren die Ergebnisse und Scores verschiedener Anfragen
- schalten von Query auf Filter Context um

Mögliche Anfragen sind…

- Constant Score Queries
- Boolean Queries
- Boosting Queries
- Disjunction Max Query
- Function Score Query

https://www.elastic.co/guide/en/elasticsearch/reference/current/compound-queries.html

# Constant Score Queries

Eine Constant Score Query definiert einen Filter Context um eine Anfrage im Query Context mit Scores

In einem Filter Context erhalten Suchtreffer standardmäßig den Score 1

Der optionale Parameter `boost` definiert einen konstanten Score für Suchtreffer

```
GET /amazon/_search
{
  "query": {
    "term": {
      "manufacturer.keyword": "victory multimedia"
    }
  }
}

GET /amazon/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "term": {
          "manufacturer.keyword": "victory multimedia"
        }
      },
      "boost": 1.2
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-constant-score-query.html

data/amazon-google-products
snippets/compound queries/constant_score_queries.txt

# Boolean Queries

Boolean Queries kombinieren Suchanfragen die auf unterschiedliche Weise in den Suchergebnissen auftreten müssen

Die Occurrence Types sind…

- `must`: Die Suchanfrage *muss* in den Suchergebnissen auftreten. Die Suchergebnisse tragen zum Score bei (Query Context)
- `should`: Die Suchanfrage *sollte* in den Suchergebnissen auftreten. Die Suchergebnisse tragen zum Score bei (Query Context)
- `filter`: Die Suchanfrage *muss* in den Suchergebnissen auftreten. Die Suchergebnisse tragen nicht zum Score bei (Filter Context)
- `must_not`: Die Suchanfrage *darf nicht* in den Suchergebnissen auftreten. Die Suchergebnisse tragen mit einem Score von 0 nicht zum Score bei (Filter Context)

```
GET /amazon/_search
{
  "query": {
    "bool": {
      "must": [...],
      "should": [...],
      "filter": [...],
      "must_not": [...]
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html

snippets/compound-queries/boolean_queries.txt

# Suche – Was ist relevant?

Beziehungen zwischen Dokumenten

# 1:n-Beziehungen

Join Fields setzen 1:n-Beziehungen (Eltern/Kind, engl. parent/child) um

Eltern- und Kindelemente müssen im gleichen Shard indexiert werden

In einem Index ist nur höchstens ein Join Field erlaubt

Ein Elter darf mehrere Kinder haben

Es darf mehrere Eltern geben

Mehrstufige Eltern-Kind-Beziehungen sind möglich, aber nicht performant

```
PUT /my-index
{
  "mappings": {
    "properties": {
      "my_id": {
        "type": "keyword"
      },
      "my_join_field": {
        "type": "join",
        "relations": {
          "question": "answer"
        }
      }
    }
  }
}
```

```
PUT my-index/_doc/1?refresh
{
  "my_id": "1",
  "text": "This is a question",
  "my_join_field": {
    "name": "question"
  }
}
```

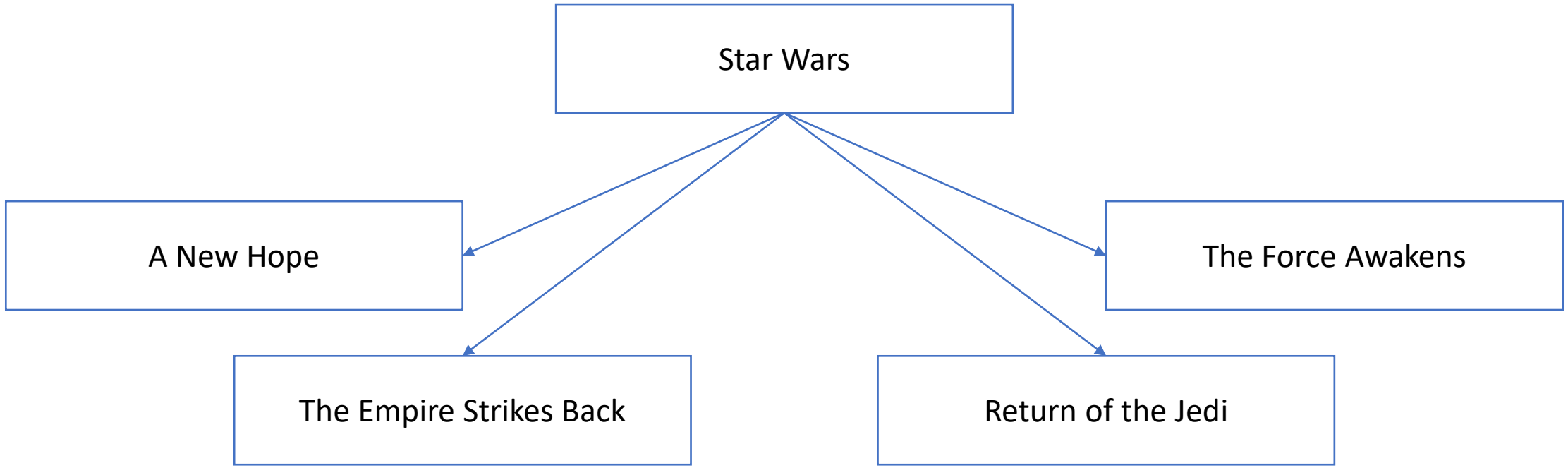
```
PUT my-index/_doc/2?routing=1&refresh
{
  "my_id": "2",
  "text": "This is an answer",
  "my_join_field": {
    "name": "answer",
    "parent": "1"
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/parent-join.html

snippets/relationships/join_fields.txt
snippets/relationships/has_child_query.txt
snippets/relationships/has_parent_query.txt

# Star Wars Franchise

Von dem Star Wars Franchise gibt es eine Reihe von Filmen




snippets/relationships/series.txt

# Suche – Was ist relevant?

Scoring

# Relevanz

Relevanz ist subjektiv

Suchalgorithmen liefern mit empirischen Methoden die Ergebnisse, die die meisten Anwender erwarten

Suchergebnisse werden in drei Schritten ermittelt, bewertet und sortiert

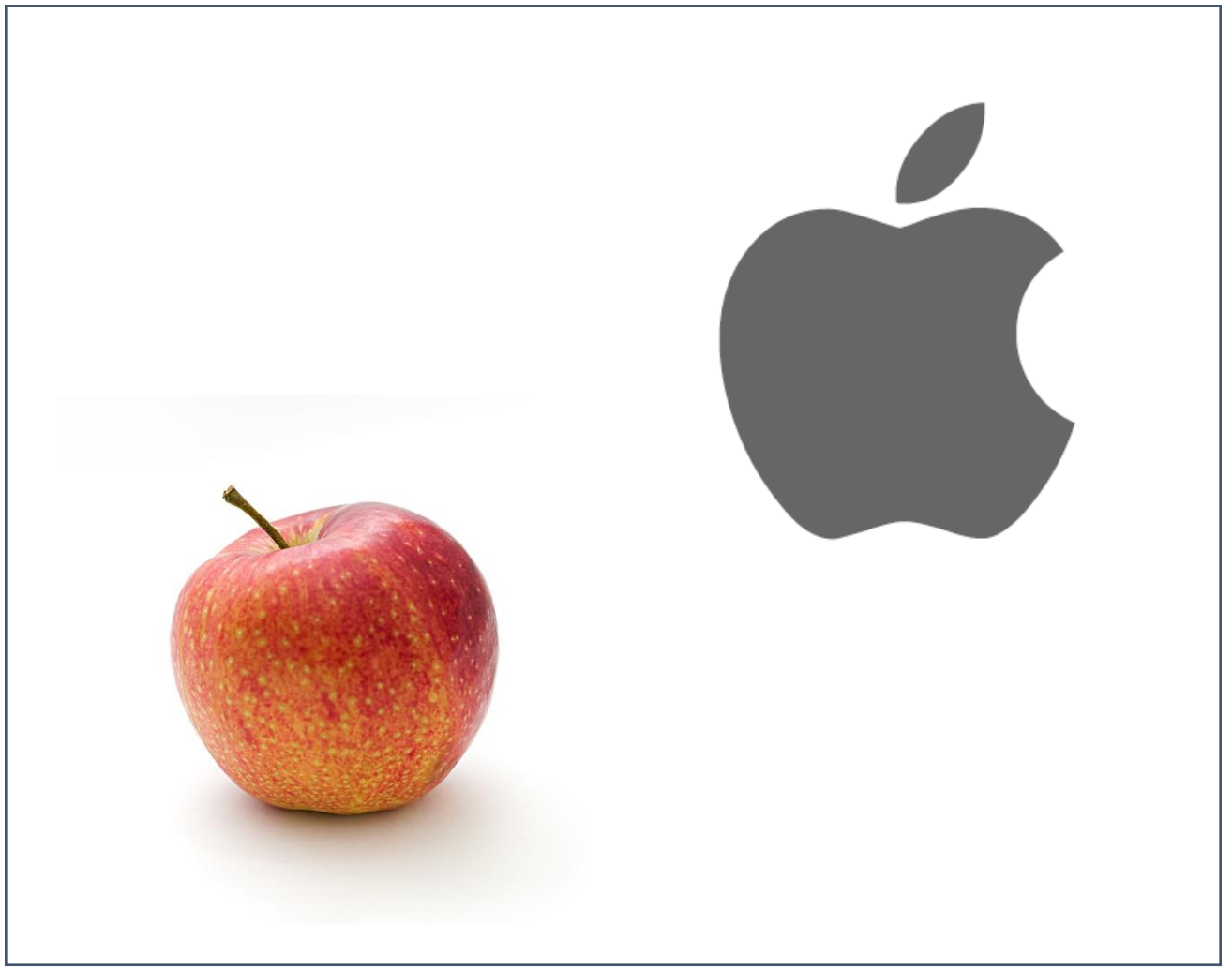
1. Finde die Dokumente die als Ergebnis in Frage kommen
2. Berechne für diese Kandidaten die Relevanz (engl. Scoring)
3. Sortiere die Ergebnisse nach deren Score

# Relevanz

## Achtung

Auch als relevant bewertete Suchtreffer sind möglicherweise nicht relevant

Auch Dokumente die nicht als relevant bewertet wurden können es sein



# Das Boolesche Modell

Ein Indexierungsterm (engl. index term) $t_i$ ist ein Ausdruck oder ein Wort

- Oft dessen reduzierte Stammform
- Zum Beispiel die Deklination von *Wortes* oder *Wörter* zu *Wort*

$T = \{t_1, t_2, \dots, t_m\}$ ist die Menge aller Terme

Eine Dokument ist jede Teilmenge von $T$

$D = \{D_1, \dots, D_n\}$ ist die Menge aller Dokumente

Eine Anfrage (engl. query) ist ein Boolescher Ausdruck in Normalform

$$Q = (W_1 \vee W_2 \vee \cdots) \wedge \cdots \wedge (W_i \vee W_{i+1} \vee \cdots)$$

$W_i$ ist true für ein $D_j$ wenn $t_i \in D_j$

Die Informationsrückgewinnung (engl. information retrieval) ist das Finden aller Dokumente, die die Anfrage $Q$ erfüllen

Finde die Dokumente in zwei Schritten

1. Finde für jedes $W_j$ in $Q$ die Menge $S_j$ an Dokumenten die $W_j$ erfüllen

   $$S_j = \{D_i | W_j\}$$

2. Die Menge aller Dokumente die $Q$ erfüllen ist dann

   $$\{S_1 \cup S_2 \cup \cdots\} \cap \cdots \cap \{S_i \cup S_{i+1} \cup \cdots\}$$

# Das Boolesche Modell

Das Boolesche Modell der Informationsrückgewinnung erfordert sehr gute Kenntnisse des eigenen Informationsbedarfs und der Dokumente

Ohne Sortierung müssen alle Ergebnisse betrachtet werden

Das Boolesche Modell liefert oft entweder sehr wenige (d.h. gar keine) oder sehr viele Ergebnisse

- AND liefert zu wenige, OR zu viele Ergebnisse

# Retrieval mit Ranking

Die Suchergebisse werden nach Relevanz sortiert (engl. ranking)

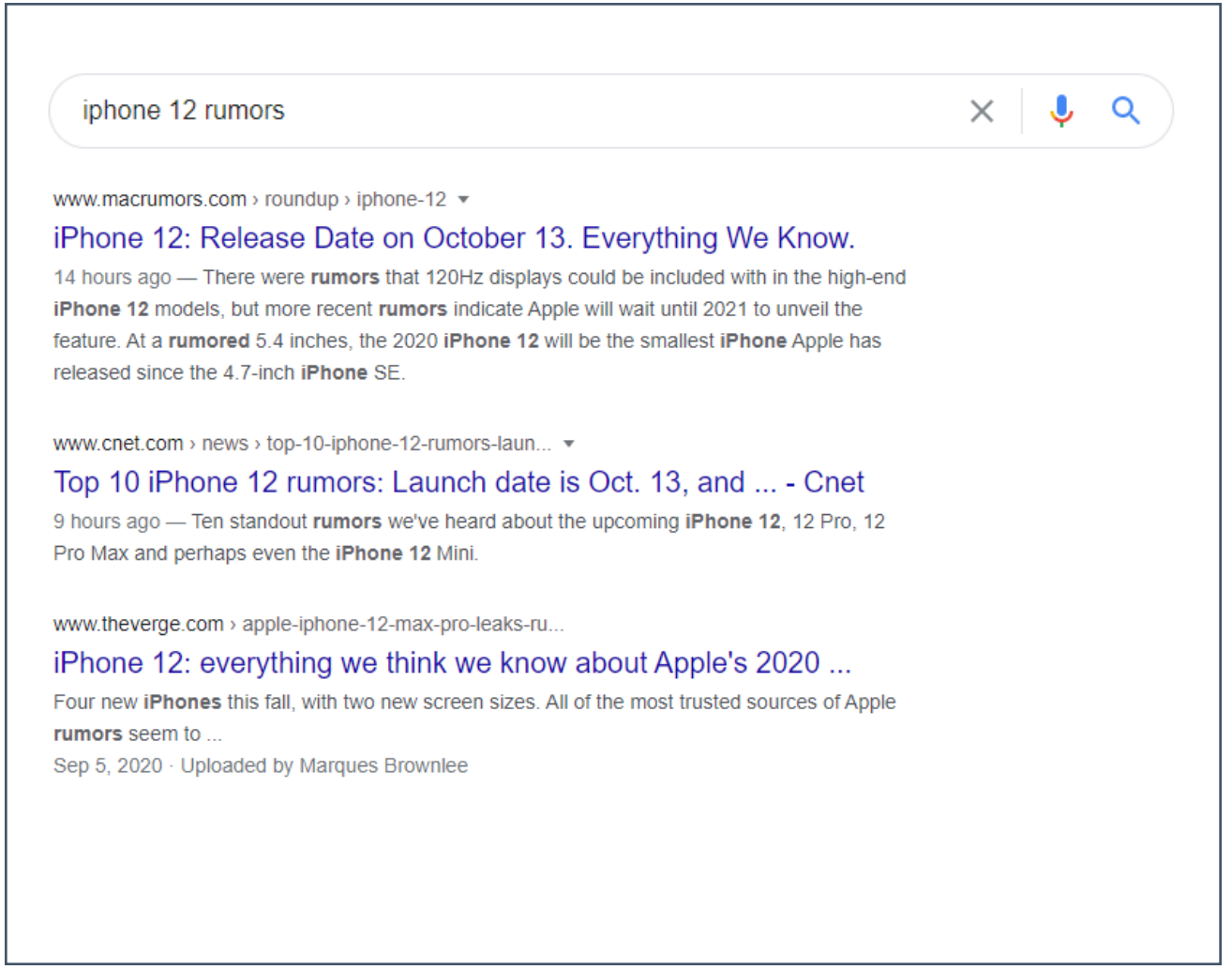
Freitextanfragen anstatt Boolescher Ausdrücke

Für eine Volltextsuche werden oft Freitextanfragen und ein Ranking kombiniert

Vorteil mit Ranking: Auch große Ergebnismengen sind kein Problem, zeige einfach nur die besten Treffer

Voraussetzung für ein Ranking: Scoring

Jedes Dokument enthält bezüglich einer Suchanfrage einen Score für seine Relevanz



# Jaccard-Koeffizient

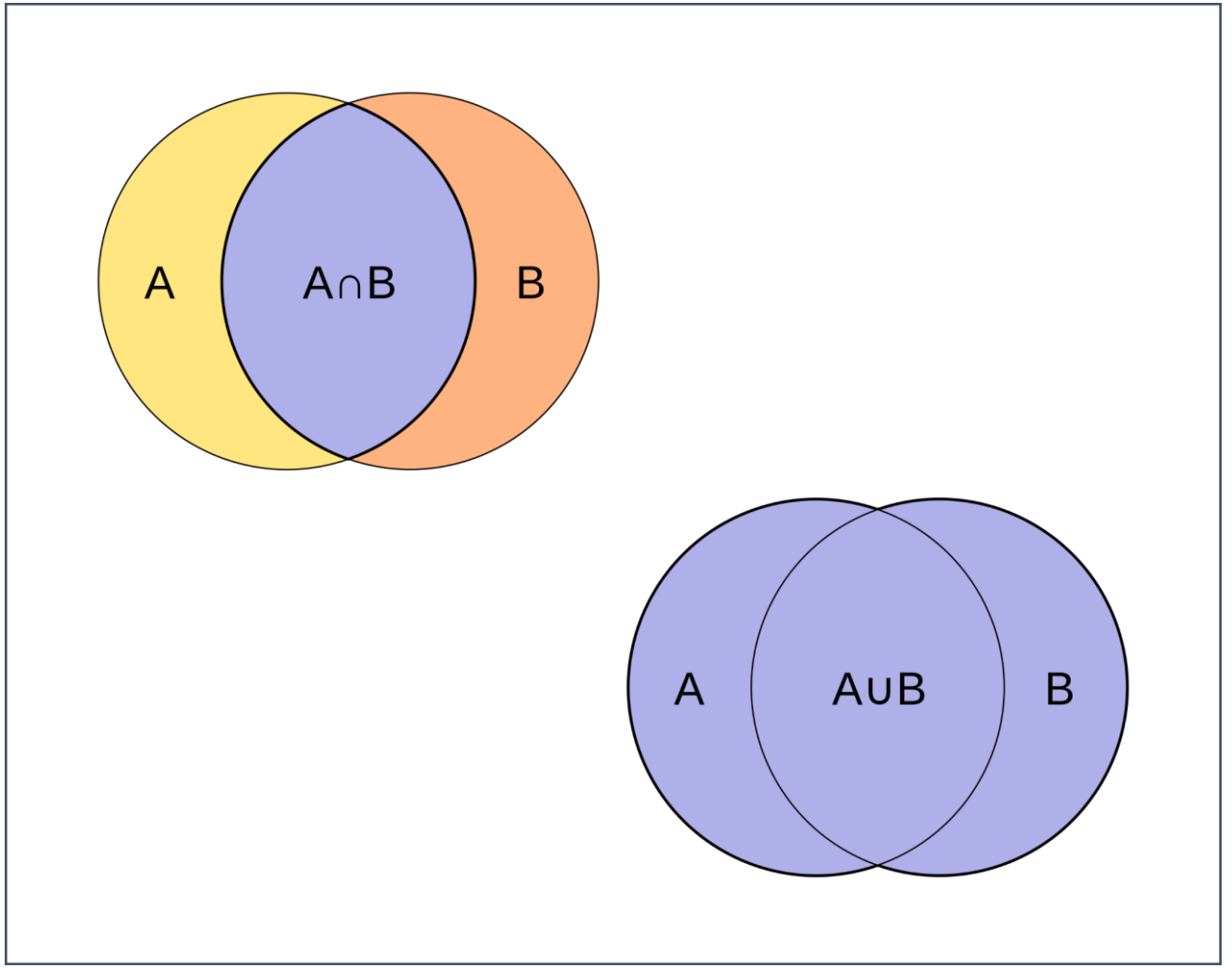
Der Jaccard-Koeffizient oder Jaccard-Index ist eine Kennzahl für die Ähnlichkeit von Mengen.

Teile die Anzahl der gemeinsamen Elemente (Schnittmenge) durch die Größe der Vereinigungsmenge

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Der Jaccard-Index berücksichtig nicht…

- Die Termfrequenz (d.h. wie oft ein Term in einem Dokument vorkommt)
- In wie vielen Dokumenten ein Term vorkommt (seltene Terme sollten höher gewichtet werden)



# TF-IDF

Die Vorkommenshäufigkeit (engl. term frequency) $\mathrm{tf}_{t,d}$ eines Terms $t$ ist die Häufigkeit von $t$ in einem Dokument $d$

Aber: Ein Dokument in dem ein Suchbegriff 10 Mal vorkommt ist nicht 10 Mal relevanter

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}\,, & \text{wenn } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{sonst} \end{cases}$$

$$\mathrm{score}(q,d) = \sum_{t \in q \cap d} \left(1 + \log \mathrm{tf}_{t,d}\right)$$

Die Dokumenthäufigkeit (engl. document frequency) $\mathrm{df}_t$ ist die Anzahl der Dokumente in denen ein Term $t$ vorkommt

$$\mathrm{df}_t = \sum_{d:t \in D} 1$$

Die inverse Dokumenthäufigkeit $\mathrm{idf}_t$ misst die Spezifizität eines Terms in der Gesamtmenge der Dokumente $D$

$$\mathrm{idf}_t = \log \frac{|D|}{\mathrm{df}_t}$$

Ein Dokument das einen seltenen Suchbegriff enthält ist relevanter, auch hier Dämpfung des Einflusses der Spezifizität

Die inverse Dokumenthäufigkeit hat nur einen Effekt bei Anfragen mit mindestens zwei Termen

# TF-IDF

Das TF-IDF-Maß ist das Produkt aus dem TF- und IDF-Gewicht

$$w_{t,d} = \log\left(1 + \mathrm{tf}_{t,d}\right) \times \log {|D|}/{\mathrm{df}_t}$$

Bezeichnungen sind auch tf.idf oder tf x idf

Der Score eines Dokuments für eine Anfrage ist

$$\mathrm{score}(q,d) = \sum_{t \in q \cap d} \mathrm{tf.idf}_{t,d}$$

# Vektorraum-Retrieval

Mit Vektorraum-Retrieval (engl. vector space model) können Suchanfragen mit mehreren Termen (engl. multi term queries) mit einem Dokument verglichen werden

Informationen werden als Punkte in einem hochdimensionalen Raum repräsentiert

Hier: Die Vektoren enthalten die TF-IDF-Gewichte der einzelnen Terme

„Alice rennt schneller als Bob“ und „Bob rennt schneller als Alice“ haben die gleichen Vektoren

Bag-of-Words-Modell: Abstraktion von der Struktur der Dokumente

# Vektorraum-Retrieval

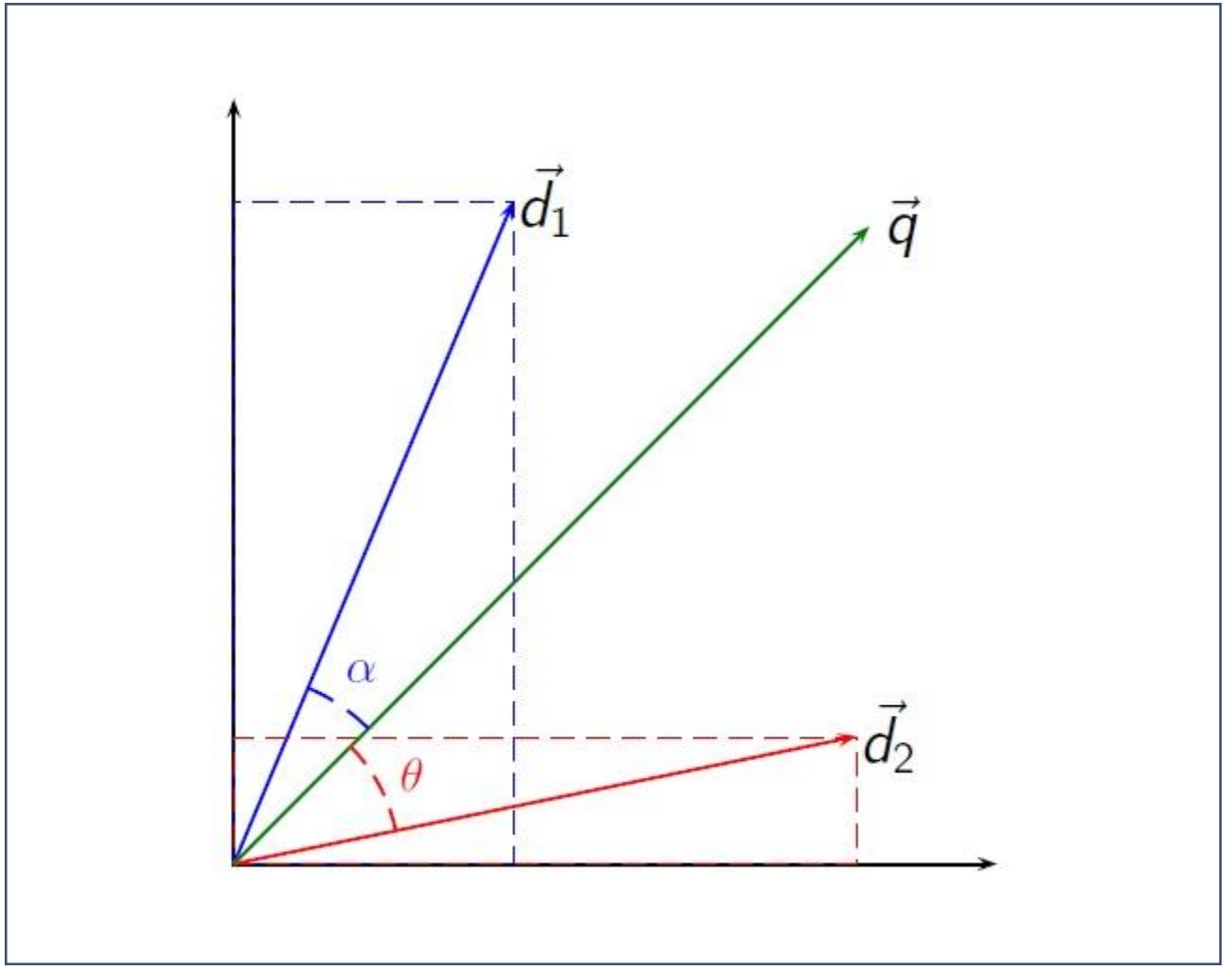
Idee 1: Dokumente *und* Anfragen als Vektoren

Idee 2: Ranking der Dokumente nach ihrer *Nähe* zur Anfrage im Vektorraum

Der euklidische Abstand als Maß für die Nähe ist keine gute Idee

- Hänge ein Dokumente $d$ an sich selbst an und nenne das neue Dokument $d'$
- Semantisch haben $d$ und $d'$ den gleichen Inhalt aber einen großen euklidischen Abstand
- Der Winkel zwischen beiden Vektoren ist 0 entsprechend der großen Ähnlichkeit

Idee 3: Ranking mit dem Winkel zwischen Dokumentvektoren und dem Anfragevektor



# Vektorraum-Retrieval

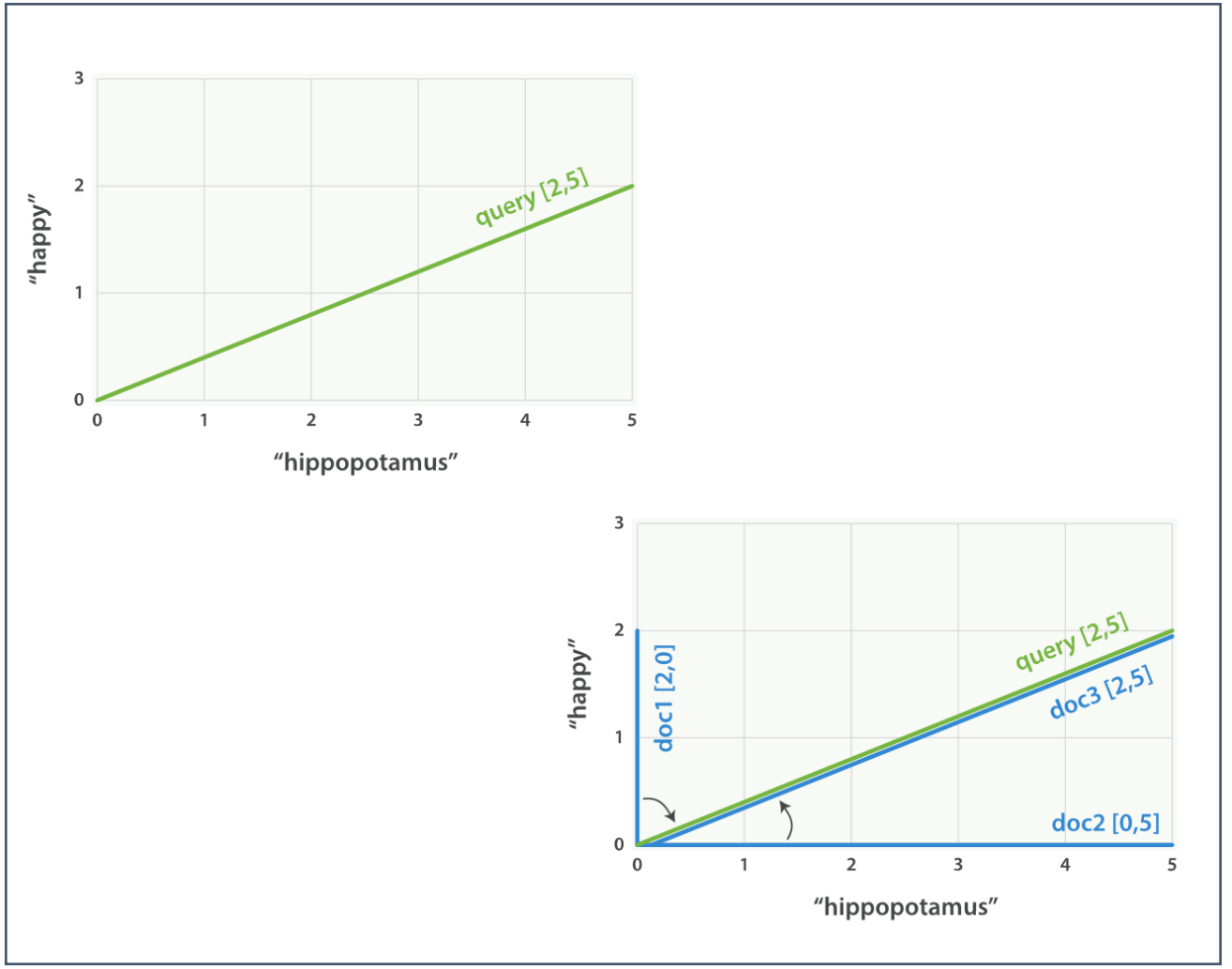
## Beispiel

Anfrage „happy hippopotamus“

Drei Dokumente

1. I am happy in summer.
2. After Christmas I’m a hippopotamus.
3. The happy hippopotamus helped Harry.

Annahme
$w_{\text{happy}} = 2$
$w_{\text{hippopotamus}} = 5$



# Kosinus-Ähnlichkeit

Der Winkel $\theta$ zwischen zwei Vektoren hängt mit dem Standardskalarprodukt zusammen

$$\mathbf{q} \cdot \boldsymbol{d} = \|\boldsymbol{q}\| \cdot \|\boldsymbol{d}\| \cos\theta$$

Die Kosinus-Ähnlichkeit von $q$ und $d$ ist der Kosinus des eingeschlossenen Winkels $\theta$

$$\cos(\boldsymbol{q}, \boldsymbol{d}) = \frac{\boldsymbol{q} \cdot \boldsymbol{d}}{\|\boldsymbol{q}\|_2 \|\boldsymbol{d}\|_2} = \frac{\boldsymbol{q}}{\|\boldsymbol{q}\|_2} \cdot \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|_2}$$

Einen Vektor kann man normieren, indem man ihn durch seine Norm teilt: $\boldsymbol{n} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$

Die Standardnorm ist

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Der normierte Vektor ist der Einheitsvektor und zeigt in die gleiche Richtung wie $\boldsymbol{v}$

Das Standardskalarprodukt zweier Einheitsvektoren ist gleich dem Kosinus des Winkels zwischen den beiden

$$\cos(\boldsymbol{q}, \boldsymbol{d}) = \boldsymbol{q} \cdot \boldsymbol{d} = \sum_{i=1}^{|V|} q_i d_i$$

# Lucene‘s Practical Scoring Function

Bis Elasticsearch Version < 5.0 der Standard-Ähnlichkeitsalgorithmus, überholt (engl. deprecated) seit Version 7.0

- $\text{tf} = \sqrt{\text{Vorkommenshäufigkeit}}$
- $\text{idf} = 1 + \ln(|D|/(\text{Dokumenthäufigkeit} + 1))$
- $\text{queryNorm}()$: 1 / Wurzel aus der Summe der quadrierten IDF-Werte der Suchterme (für eine Vergleichbarkeit von Suchanfragen)
- $\text{coord}()$: Anzahl Terme aus der Suchanfrage in einem Dokument geteilt durch die Anzahl der Terme in der Suchanfrage
- $t.\text{getBoost}()$: (Prozent-)zahl um einzelne Terme der Suchanfrage höher zu gewichten
- $\text{norm}() = 1/\sqrt{\text{Anzahl Terme im Suchfeld}}$ kombiniert mit „Field-level boost“

$$\text{score}(q,d) = \text{queryNorm}(q) \times \text{coord}(q,d) \times \sum \big(\text{tf}(t \text{ in } d) \times \text{idf}(t)^2 \times t.\text{getBoost}() \times \text{norm}(t,d)\big)(t \text{ in } q)$$

# Okapi BM25

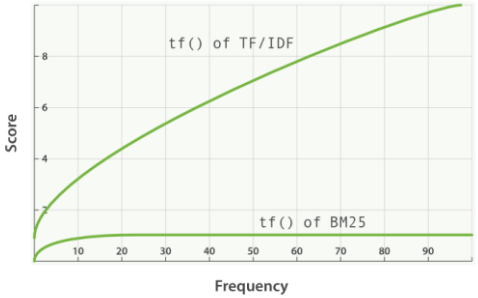
Schwäche TF-IDF: Ein Dokument das 200 Mal einen Suchbegriff enthält ist nicht doppelt so relevant wie ein Dokument mit 100 Vorkommen

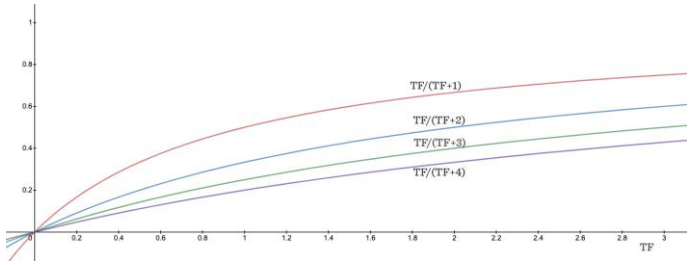
Idee: „Term saturation“ $\mathrm{TF}/(\mathrm{TF}+k)$

$q_i$ ist der i-te Term in der Suchanfrage

$f(q_i, d)$ ist die Vorkommenshäufigkeit von Term $q_i$ in Dokument $d$

$\mathrm{IDF}(q_i)$ ist die inverse Dokumentfrequenz von $q_i$: $\ln\left(1+\frac{\mathrm{docCount}-f(q_i)+0.5}{f(q_i)+0.5}\right)$




$$\mathrm{score}(q,d)=\sum_{i}^{n}\mathrm{IDF}(q_i)\times\frac{f(q_i,D)\times(k_1+1)}{f(q_i,D)+k_1\times\left(1-b+b\times\frac{\mathrm{fieldLen}}{\mathrm{avgFieldLen}}\right)}$$

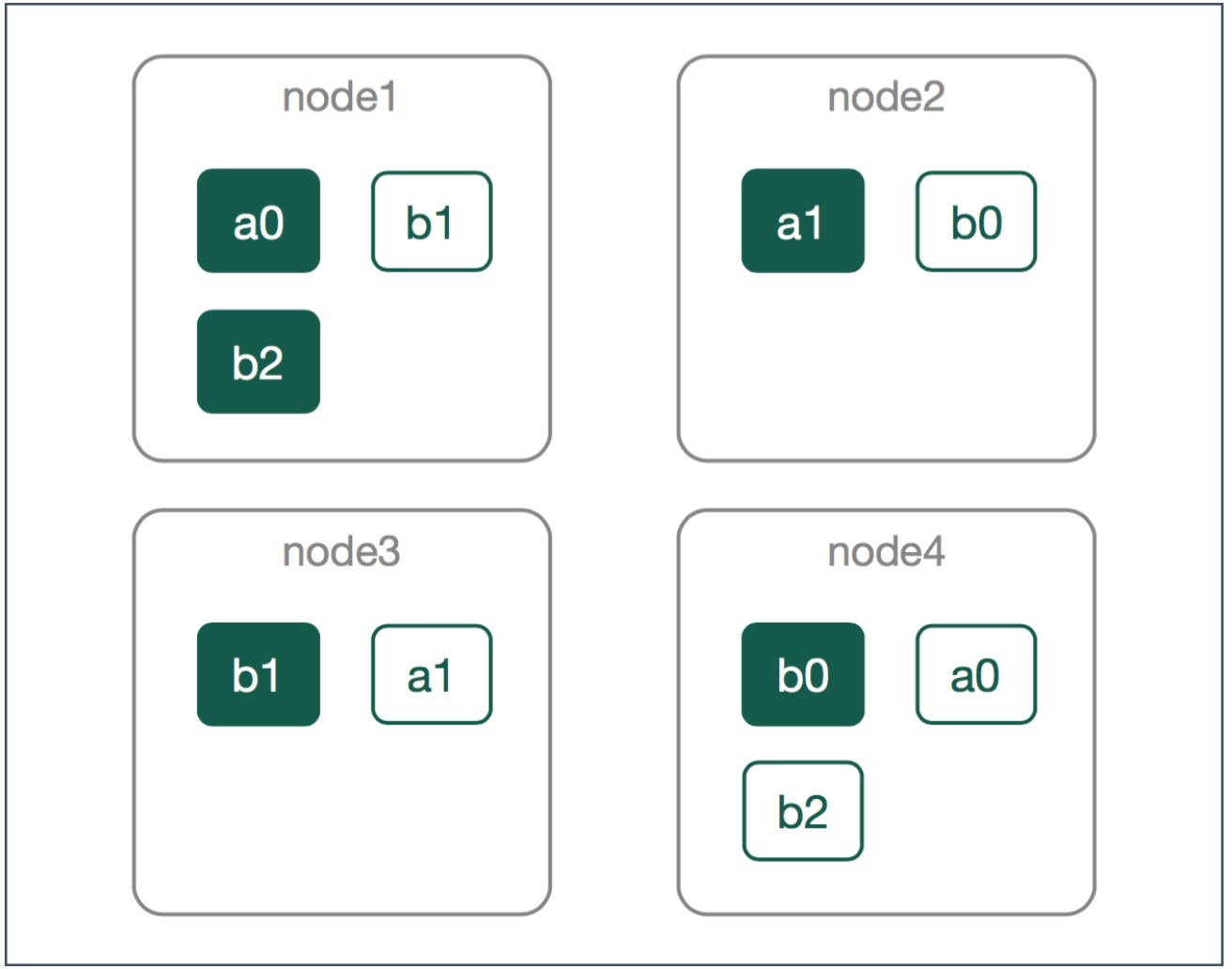
# Scoring mit Shards

Standardmäßig berechnet Elasticsearch die Scores von Suchtreffern pro Shard

Bei zu wenigen und zu unterschiedlichen Dokumenten sind die Scores nicht akkurat

## Lösungen

- Je mehr Dokumente in den Index geladen werden desto genauer werden die Scores
- Verwende mit `number_of_shards = 1` nur einen Shard (nicht empfohlen!)
- Mit der Such-Option `?search_type=dfs_query_then_fetch` werden die Vorkommenshäufigkeiten und Dokumentfrequenzen in einem extra Schritt über alle Shards hinweg abgefragt (extra Aufwand)




https://www.elastic.co/blog/practical-bm25-part-1-how-shards-affect-relevance-scoring-in-elasticsearch
https://www.elastic.co/blog/understanding-query-then-fetch-vs-dfs-query-then-fetch

DH DUALE
SH HOCHSCHULE SH

# Bildnachweis

Learning Elastic Stack 7.0: Distributed search, analytics, and visualization using Elasticsearch, Logstash, Beats, and Kibana, Pranav Shukla, Sharath Kumar M N, Packt Publishing; 2nd revised edition, 2019

https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up

https://en.wikipedia.org/wiki/Jaccard_index#/media/File:Intersection_of_sets_A_and_B.svg

https://en.wikipedia.org/wiki/Jaccard_index#/media/File:Union_of_sets_A_and_B.svg

https://en.wikipedia.org/wiki/Vector_space_model#/media/File:Vector_space_model.jpg

https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html

https://www.elastic.co/blog/practical-bm25-part-2-the-bm25-algorithm-and-its-variables

http://www.kmwllc.com/index.php/2020/03/20/understanding-tf-idf-and-bm25/

https://www.elastic.co/blog/every-shard-deserves-a-home

https://en.wikipedia.org/wiki/Geohash#/media/File:Geohash-OddEvenDigits.png