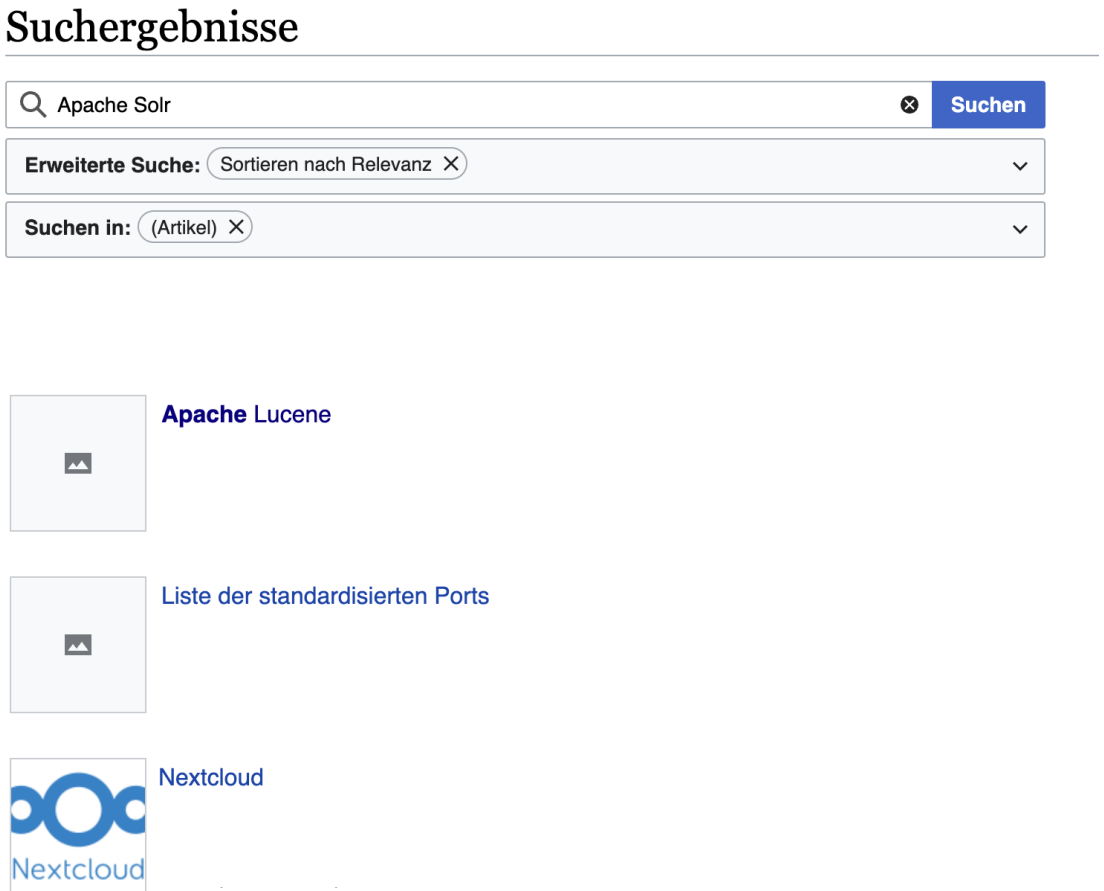
Apache Solr

Wikipedia



- Nur Titel durchsuchen reicht nicht
  - Apache Solr hat keinen eigenen Wikipediaartikel
- Anforderungen an die Suche:
  - Text sollte durchsuchbar sein
  - Suchergebnisse sollten möglichst relevant sein

Volltextsuche

- Intuitiv: Wort für Wort nach dem Suchbegriff suchen
  - Worst Case Laufzeit 1 Artikel mit n Wörtern: O(n)
- Index: Volltext wird in Einheiten wie z.B. Worte zerlegt
  - Wort assoziiert mit Artikeln, die dieses Wort enthalten

1. Google ist eine Suchmaschine
2. Suchmaschinen durchsuchen Dokumente
3. Dokumente haben eine Struktur

Wort	Dokumente
Google	1
Suchmaschine	1, 2
durchsuchen	2
Dokumente	2, 3
Struktur	3

Volltextsuche

- Suchbegriff wird im Index gesucht
  - Verweise auf Dokumente, die den Suchbegriff beinhalten
- Index kann schnell durchsucht werden
  - z.B. Verwenden einer HashMap: O(1)

1. Google ist eine Suchmaschine
2. Suchmaschinen durchsuchen Dokumente
3. Dokumente haben eine Struktur

Wort	Dokumente
Google	1
Suchmaschine	1, 2
durchsuchen	2
Dokumente	2, 3
Struktur	3

Ranking

- Relevante Suchergebnisse sollten oben stehen
- Sortieren nach Relevanz
  - Anzahl der Vorkommnisse des Suchbegriffs
  - Position Suchbegriff im Artikel
- Berechnen eines Scores

Verteilte Indizes

- bei sehr großen Suchindizes → Index verteilen
- Shards: jeder Shard enthält einen Teil des Suchindex
- Replica: schaffen Redundanz
  - alle Replica eines Shards enthalten dieselben Daten
  - Leader: übernimmt Änderungen am Shard zuerst, andere Replica übernehmen Änderungen dann nach und nach

Shards: Pro und Contra

- Vorteile:
  - Weniger Speicherverbrauch pro Server
  - Parallelisieren der Suche
    - es muss pro Server nur noch ein Teil des Gesamtindex durchsucht werden
- Nachteile:
  - höherer Verwaltungsaufwand: zu welchem Shard wird ein neuer Eintrag hinzugefügt

Wort	Dokumente
Google	1
Suchmaschine	1, 2
durchsuchen	2
Dokumente	2, 3
Struktur	3

Shard 1:

Wort	Dokumente
Google	1
Suchmaschine	1, 2
Dokumente	2, 3

Shard 2:

Wort	Dokumente
durchsuchen	2
Struktur	3

Replica: Pro und Contra

- Vorteile:
  - Ausfallsicherheit: wenn eine Replik ausfällt, ist der Shard trotzdem noch verfügbar
  - Lastenverteilung: viele Suchanfragen können auf die Replica verteilt werden
- Nachteile:
  - Änderungen werden nach und nach übernommen: inkonsistente Daten je nach Replik
  - Verwaltungsaufwand: Synchronisation der Replica ist aufwändig

Shard 1 Replica 1

Wort	Dokumente
Google	1
Suchmaschine	1, 2
Dokumente	2, 3

Shard 1 Replica 2

Wort	Dokumente
Google	1
Suchmaschine	1, 2
Dokumente	2, 3

Solr

- Open-Source Suchserver der Apache Foundation
- bietet eine REST API zum Durchsuchen und indizieren von Daten
- SolrCloud: verteilte Suchindizes über Shards und Replica
- basiert auf Apache Lucene: Java Bibliothek für Suchfunktionalitäten

Quelle: https://commons.wikimedia.org/wiki/File:Apache_Solr_logo.svg

Indexing in Solr

- Solr Dokumente bestehen aus Feldern
  - können als XML oder JSON an die REST API gesendet werden
  - Bibliotheken in vielen Programmiersprachen (z.B. SolrJ in Java und Kotlin)
  - Solr Cell mit Apache Tika: PDFs, Word Dokumente und andere Formate indizieren
- Schema: beschreibt welche Felder mit welchen Datentypen Dokumente enthalten

Suche in Solr

q=title:apache

- vor : → Feld, in dem gesucht werden soll
- nach : → Suchbegriff, nach dem gesucht werden soll

fl=title

- es sollen nur die Titel der Suchergebnisse zurückgegeben werden

start=0&rows=10

- zeige die Suchergebnisse 0 bis 10

SolrCloud

- Skalierung von Solr Anwendungen
- ermöglicht Shards und Replica
- zentrale Instanz von Apache ZooKeeper
  - verwaltet Nodes und Konfigurationen
  - koordiniert Änderungen aller Nodes bei Änderungen an der Kofiguration
- Collections: fassen alle Shards und Replica eines Suchindex zusammen

Demo

0. Demo der Demo App

1. SolrCloud Server über Docker aufsetzen - Portainer

2. Solr Dokumente indizieren - SolrJ

1. Verbindung zur Solr Instanz herstellen

```
solr = HttpSolrClient.Builder(urlString).build()
```

2. Dokument erstellen:

```
fun toSolrDocument(): SolrInputDocument? {
        val solrDocument = SolrInputDocument()
        solrDocument.addField("url", this.url)
        solrDocument.addField("lastModified", this.lastModified)
        solrDocument.addField("title", this.title)
        solrDocument.addField("headlines", this.headlines)
        solrDocument.addField("paragraphs", this.paragraphs)

        return solrDocument
}
```

2. Solr Dokumente indizieren - SolrJ

3. Dokument zum Index hinzufügen:

```
solr.add(siteContents.toSolrDocument())
```

4. Änderungen commiten:

```
solr.commit()
```

3. Index durchsuchen - SolrJ

```
fun searchSolr(includeParagraphs: Boolean, searchString: String): SolrDocumentList {
        val queryParamMap: MutableMap<String, String> = HashMap()

        if (includeParagraphs) queryParamMap["q"] = "title:$searchString OR
                paragraphs:($searchString)"
        else queryParamMap["q"] = "title:$searchString"

        queryParamMap["fl"] = "title, url"
        queryParamMap["rows"] = "5000"

        val queryParams = MapSolrParams(queryParamMap)
        return solr.query(queryParams).results
}
```

Quellen

- siehe Ausarbeitung zu Apache Solr