Solr - database & search engine

Paweł Pyciński & Arkadiusz Pospieszny

Jagiellonian Uniwersity

Table of contents

Section 1

SOLR - Teoria

Solr to samodzielny serwer wyszukiwania korporacyjnego z interfejsem API podobnym do REST.

Definition

Solr jest wysoce niezawodny, skalowalny i odporny na awarie, oferując rozproszone indeksowanie, replikację i zapytania z równoważeniem obciążenia, automatyczne przełączanie awaryjne i odzyskiwanie, scentralizowaną konfigurację i wiele więcej. Solr zasila funkcje wyszukiwania i nawigacji wielu największych witryn internetowych na świecie.

Zalety korzystania z Solr:

- Ogromna baza wiedzy tworzona przez użytkowników
- Obszerna dokumentacja z przykładami
- Wsparcie dla Docker
- Wysoka skalowalność oparta na ZooKeeper
- Dokumenty mogą być przechowywane w formatach JSON, XML, CSV lub binarnych, a zapytania można wykonywać w tych samych formatach

Zookeeper

Apache ZooKeeper to rozproszony system zarządzania konfiguracją i synchronizacji, który jest używany do zarządzania wieloma systemami rozproszonymi. ZooKeeper jest ważnym elementem w zarządzaniu klastrami Solr. Kluczowe cechy ZooKeeper:

- **Zarządzanie konfiguracją**: Przechowuje konfiguracje systemów rozproszonych w sposób scentralizowany.
- **Synchronizacja usług**: Umożliwia synchronizację usług w klastrach rozproszonych.
- **Nadmiarowość**: Zapewnia wysoką dostępność poprzez replikację danych.
- **Niskie opóźnienia**: Zapewnia niskie opóźnienia w dostępie do danych konfiguracyjnych.

Lucene

Apache Lucene to biblioteka open-source, która jest napisana w Java i umożliwia pełnotekstowe indeksowanie i wyszukiwanie dokumentów Apache Solr i analizę tekstu.

Indeksowanie

1. Tworzenie dokumentu
2. Analizator tekstu - tokenizacja i filtrowanie (np: usuwanie stop-wordów), tokeny zapisują odwrócone indeksy, które umożliwiają szybsze przeszukiwanie dokumentów.
3. Dodanie dokumentu do indeksu - Lucene przechowuje indeks w formie segmentów na dysku.

```
Document doc = new Document();
doc.add(new TextField("content", "Kot lubi bawić się kłębkiem wełny.", Field.Store.YES));

Analyzer analyzer = new StandardAnalyzer();

IndexWriterConfig config = new IndexWriterConfig(analyzer);
IndexWriter writer = new IndexWriter(indexDirectory, config);
writer.addDocument(doc);
writer.close();
```

Wyszukiwanie

1. Zapytanie - obiekt Query oferuje typy zapytań jak: TermQuery (szukanie pojedynczych terminów), PhraseQuery (szukanie fraz), BooleanQuery (łączące różne zapytania logicznie)
2. Analiza
3. Przeszukiwanie i zwracanie

```
String queryStr = "kot bawić";
Query query = new QueryParser("content", analyzer).parse(queryStr);

IndexReader reader = DirectoryReader.open(indexDirectory);
IndexSearcher searcher = new IndexSearcher(reader);
TopDocs results = searcher.search(query, 10);

for (ScoreDoc scoreDoc : results.scoreDocs) {
Document doc = searcher.doc(scoreDoc.doc);
System.out.println("Found: " + doc.get("content"));
}
reader.close();
```

1. Tokenizacja - Tokenizer (dzieli tekst na tokeny na podstawie określonych reguł)
2. Filtry tokenów

```
Analyzer customAnalyzer = new Analyzer() {
        @Override
        protected TokenStreamComponents createComponents(String fieldName) {
        Tokenizer source = new StandardTokenizer();
        TokenStream filter = new LowerCaseFilter(source);
        filter = new StopFilter(filter, StopAnalyzer.ENGLISH_STOP_WORDS_SET);
        filter = new PorterStemFilter(filter);
        return new TokenStreamComponents(source, filter);
        }
};
```

Interfejs użytkownika Solr jest bardzo prosty i nie oferuje wielu funkcji. W instalacji Docker mamy:

1. **Dashboard** - przegląd systemu i JVM
2. **Logging** - Rejestrowanie złych żądań od klientów oraz wewnętrznych ostrzeżeń Java w Solr
3. **Security** - Materializacja interfejsu naszego pliku security.json, który jest tworzony po uruchomieniu kontenera. Tutaj możemy również modyfikować opcje zabezpieczeń.
4. **Core Admin** - Szczegółowe informacje o rdzeniach w systemie - tutaj możemy również tworzyć rdzeń, ale musimy mieć już gotowe pliki konfiguracyjne.
5. **Java Properties** - ustawienia Java
6. **Thread Dump** - aktualny zrzut wątków
7. **Core slider** - tutaj możesz wybrać rdzeń, z którym chcesz się komunikować za pośrednictwem interfejsu użytkownika. Po tym masz zakładki do pracy z nim, takie jak Zapytania, Dokumenty, gdzie możesz wysyłać zapytania do indeksu lub przesyłać dokumenty.

Solr Schema - solrConfig.xml

solrconfig.xml to jeden z najważniejszych plików konfiguracyjnych w Apache Solr. Definiuje ustawienia i zachowania instancji Solr, takie jak sposób indeksowania, przetwarzanie zapytań oraz konfigurację różnych komponentów Solr.
solrconfig.xml zawiera konfiguracje, które kontrolują:

- **Zarządzanie dokumentami** - Jak dokumenty są dodawane, aktualizowane i usuwane z indeksu.
- **Przetwarzanie zapytań** - Jak zapytania są obsługiwane, jakie filtry i analizy są stosowane.
- **Cache** - Konfiguracja pamięci podręcznej (cache), która wpływa na wydajność wyszukiwania.
- **Pluginy i komponenty** - Dodawanie i konfiguracja różnych pluginów i komponentów, takich jak analiza tekstu, faceting, itp.
- **Logowanie** - Ustawienia dotyczące logowania i poziomów szczegółowości logów.

Solr Schema - solrConfig.xml

W Apache Solr, dane są zorganizowane w strukturze zwanej schematem. Każdy dokument w Solr ma różne pola, takie jak tytuł, autor, data publikacji, kategoria itp. Te pola są zdefiniowane w pliku schema.xml,a do tych danych odnosi się wspomniany wcześniej sorlConfig.xml

Dlaczego to jest ważne?

- **Precyzyjne przetwarzanie zapytań** - Solr musi wiedzieć, które pola przeszukiwać, aby znaleźć odpowiednie wyniki.
- **Skuteczna analiza tekstu** - Niektóre komponenty, jak analiza tekstu, muszą wiedzieć, które pola analizować.
- **Faceting i filtrowanie** - Solr musi wiedzieć, które pola użyć do grupowania wyników.
- **Optymalizacja wydajności** - Wiedza o polach pomaga w optymalizacji działania Solr, na przykład poprzez ustawienia cache.

Replicas, Shards

Jak działają shardowanie i replikowanie w Solr?

Shardowanie:
Dane są dzielone na shard podczas indeksowania. Solr automatycznie określa, który shard powinien przechowywać dane na podstawie kluczy podziału (shard key). Gdy zapytanie jest wysyłane do Solr, jest ono rozdzielane na poszczególne shardy. Każdy shard przetwarza swoją część zapytania równolegle, a wyniki są następnie łączone w celu uzyskania ostatecznej odpowiedzi.

Replikowanie:
Podczas indeksowania dane są nie tylko zapisywane na głównym shardzie (primary shard), ale również kopiowane do replik (replica shards). Repliki są aktualizowane na bieżąco, aby zapewnić, że mają te same dane co główny shard. Zarówno główny shard, jak i jego repliki mogą odpowiadać na zapytania, co zwiększa wydajność i zapewnia wysoką dostępność.

Scaling Solr

Rozproszenie Solr (SolrCloud) umożliwia skalowanie wyszukiwania i indeksowania poprzez podział danych na wiele serwerów. Wprowadzenie SolrCloud pozwala na zarządzanie dużymi zbiorami danych, zapewnienie wysokiej dostępności oraz zwiększenie wydajności.
Główne elementy SolrCloud

1. **ZooKeeper** - Centralny system zarządzania konfiguracją i synchronizacji dla SolrCloud. Przechowuje informacje o stanie klastrów, shardów i replik.
2. **Shardy** - Fragmenty indeksu, które przechowują część danych. Każda kolekcja w SolrCloud jest podzielona na shardy.
3. **Repliki** - Kopie shardów przechowywane na różnych serwerach w celu zapewnienia wysokiej dostępności i równoważenia obciążenia.

AutoCommit - performance improvement

Solr obsługuje dwa rodzaje commitów: twarde commity (hard commits) i miękkie commity (soft commits):

- **Hard commit** wywołuje fsync na plikach indeksu, aby upewnić się, że zostały zapisane na stabilnym nośniku (dysku). Bieżący dziennik transakcji jest zamykany i otwierany jest nowy. Opcjonalnie twardy commit może również udostępnić dokumenty do wyszukiwania, ale może to nie być idealne w niektórych przypadkach, ponieważ jest bardziej kosztowne niż miękki commit. Domyślnie działania commitowania prowadzą do twardego commitu wszystkich plików indeksu Lucene na stabilnym nośniku (dysku).
- **Soft commit** jest szybszy, ponieważ tylko udostępnia zmiany w indeksie i nie wykonuje fsync na plikach indeksu, nie rozpoczyna nowego segmentu ani nowego dziennika transakcji. Kolekcje wyszukiwania, które mają wymagania dotyczące NRT (Near Real-Time), będą chciały wykonywać miękkie commity wystarczająco często, aby spełnić wymagania aplikacji dotyczące widoczności. Miękki commit może być "mniej kosztowny" niż twardy commit (przy openSearcher=true), ale nie jest darmowy. Zaleca się ustawienie tego na jak najdłuższy czas, jaki jest rozsądny w kontekście wymagań aplikacji.

więcej

Section 2

Execices

Preparation

It is good to have a Postman App - it will be easier to send requests to solr, of course it also could be done using curl.

First of all we need to setup SOLR, since we want to do this we have to possiblities:

1. Solr as standalone service in Windows / Linux / MacOs
2. Solr in docker

Repository with course files link

Solr Auth

We have some possibilities according to documentation:

- Basic Authentication Plugin
- Kerberos Authentication Plugin (a protocol for authenticating service requests between trusted hosts across an untrusted network)
- JWT Authentication Plugin
- Certificate Authentication Plugin (extracting the user principal out of the client's certificate)
- Hadoop Authentication Plugin

In this tutorial we use Basic Auth, with role based authorization. File **security.json** contains users and user privileges. But wahat if we want to create a user with non standard password? We need to use third party software

Collection & Schema vs Cores

Solr differs depends on which implementation we choose. We can choose Standalone solr instalation where we have Collections and Schemas.

- **Schema** - it is a template which contains fields definitions, libraries which are used to search, analyzers and finally fields.
- **Collection** is created based on schema. Multiple collections can have same schema

Solr in Container differs, it has **Core** which is actually merge of Schema & Collection in one object.

Fields

A field type defines the analysis that will occur on a field when documents are indexed or queries are sent to the index. In Solr different we can define fields which our data supposed to have. By default it is enabled field guessing where if you submit document with new field it adds it with guessed type to solrconfig.xml. We can disable this feauture click

Commony used field types:

- pint
- string
- pdate
- text_general
- _nest_path_
- text_pl and other languages text_en, text_ru

and some more click . . . We can also create our own types of fields click

CopyFields & Dynamic Fields

Copying Fields You might want to interpret some document fields in more than one way. Solr has a mechanism for making copies of fields so that you can apply several distinct field types to a single piece of incoming information.

<copyField source="cat" dest="text" maxChars="30000" />

A common usage for this functionality is to create a single "search" field that will serve as the default query field when users or clients do not specify a field to query.

More info

Dynamic fileds

Dynamic fields allow Solr to index fields that you did not explicitly define in your schema.

This is useful if you discover you have forgotten to define one or more fields. Dynamic fields can make your application less brittle by providing some flexibility in the documents you can add to Solr.

<dynamicField name="*¡" *type* = " *int*" *indexed* = " *true*" *stored* = " *true*" / >

Fields - exercise

1. Add fields to Client Schema:
   - 1.1 **name** - string (required) (indexed, stored, docValues)
   - 1.2 **last_name** - string (required) (indexed, stored, docValues)
   - 1.3 **birth_date** - pdate (indexed, stored)
   - 1.4 **description** - text_en (indexed, stored)
   - 1.5 **age** - pint (indexed, stored)
   - 1.6 **salary** - pdouble (indexed, stored)
   - 1.7 ***_additionalInfo** -dynamic field - text_en (indexed, stored)
   - 1.8 **cars** - string (indexed, stored, multiValued)
   - 1.9 **personalData** -copy field - string (indexed, stored, multivalued) (contains name,last_name,age,salary)

fields params description

filed add guide

Json for Orders collection

Solution

Nested documents

In Solr we have possibility to store in our index nested jsons which is something similar to join operations in SQL

```
{
    "name": "jonh",
    "last_name": "smith",
    "age":30
    "children":{[
        child:{
            "child_name:"max",
            "child_age":18
        },
        child:{
            "child_name:"joanna",
            "child_age":2
        }]
    }
}
```

Nested documents fields - req

```
<field name="_root_" type="string" indexed="true" stored="false"
docValues="false" />
```

we need to have clear index without data while adding data. From SOLR 9 this filed is auto added. We also need fields:

```
<fieldType name="_nest_path_" class="solr.NestPathField" />
<field name="_nest_path_" type="_nest_path_" />
<field name="_nest_parent_" type="string" indexed="true"
stored="true" />
```

Solr automatically populates this field in child documents but not root documents.

1. All field names in the schema can only be configured in one different types of child documents
2. It may be infeasible to use required for any field names that aren't required for all types of documents.
3. Even child documents need a globally unique id.

Nested documents - exercise

From SOLR 9 this fields are added by default to index, all we need is to add fileds for our children

1. Add nested field to Clients Schema with following fields:
   1.1 *string* addreses (stored, indexed)
   1.2 *string* address_line_1 (stored, indexed)
   1.3 *string address*_line_2 (stored, indexed)
   1.4 *text_en* note (stored, indexed)
2. Add document with child
3. Add document without child then add it using atomic upadate
4. Modify latest added document replace **note**
5. delete latest child

Querying results

Querying rules:

- **q** - Query part
- **q.op** - Operator between fq conditions
- **sort** - sorting - (score desc, price desc)
- **qf** - additional conditions (not influence on score)
- **start, rows** - pagination
- **fl** - filed list (id,name,note,addresses[child])
- **df** - default search field
- **paramset(s)** - debug params
- **wt** - return response type (json, xml, etc. . .)
- **indent on** - debug param
- **defType** - query parser mode
- **stopWords** - include stop words (in txt file, words which enginge ommits to get better results)

more info

Search - exercises

We already searched for data when we added new stuff to SOLR. Now, based on previous knowledge lets make some harder execise: We will Query **Orders** collection

1. Query elements which are created by any John in two ways. (bonus try weigths)
2. Query elements which not are created by any John in two ways.
3. Query for elements which are created by John and has nice stuff in order_description
4. Query for element which is never than 7 years

Search - nested documents

As we can see previously to see in a proper way child docs we need to add *,[child] in fl field.

1. Query for all docs with proper child display
2. Query only children for parent 1_1
3. Query for all matching parents where in children has phrase **234** in any address field, return all children for parents
4. Query for all matching parents where in children has phrase **234** in any address field, return only matching children
5. Query for all matching parents where name is **Bill** and in children has phrase **234** in any address field, return all children for parents

Querying fields

We call api to check if filed is already created and working

```
http://localhost:8983/solr/Orders/schema/fields
```

with body - api to add fields

Facets, highilhts

Highilghing gives us extra docs in our result query which shows where solr found search phrase. in solr UI we need to enable **hl** and add to **hl-fl** fields where we searched to find phrase.

more info

Execise

1. find all orderes where order_details has Szczecin and highlight where result was found

Resources

- solr.apache.org

The End