01-2014 | Frühling | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

Java hebt ab

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

4 191978 304903 01

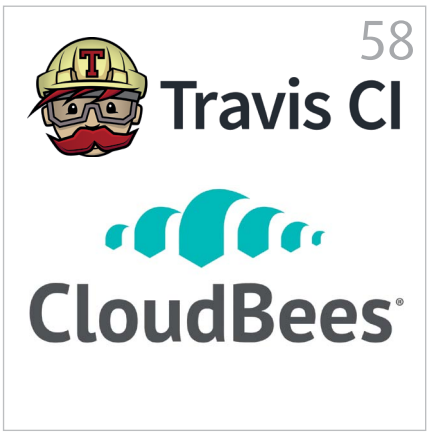
iJUG Verbund

Sonderdruck

Java macht wieder richtig Spaß:
Neuigkeiten von der JavaOne, Seite 8

Interview mit Mark Little über das Wachstum und die Komplexität von Java EE 7, Seite 30

JavaFX oder eine HTML5-basierte Technologie:
Wo und warum der Einsatz von JavaFX sinnvoll ist , Seite 44



Cloud-hosted Continuous Integration mit CloudBees und Travis CI, Seite 58

Solr und ElasticSearch – Lucene on Steroids

Florian Hopf, Freiberuflicher Softwareentwickler

Lucene ist der De-facto-Standard für die Implementierung von Suchlösungen auf der JVM. Statt Lucene direkt zu nutzen, werden vermehrt die darauf aufbauenden Suchserver Apache Solr und ElasticSearch eingesetzt. Der Artikel stellt diese vor.

In der letzten Ausgabe haben wir gesehen, wie man mit der Bibliothek Lucene das Grundgerüst einer flexiblen Suchlösung implementiert. Lucene kümmert sich um die Speicherung und das Auslesen von Textinhalten aus einem invertierten Index. Das Verhalten der Suche wird maßgeblich durch den Analyzing-Prozess beeinflusst. Vorgefertigte oder eigene Analyzer überführen dabei die Textinhalte in die im Index gespeicherten Terme. Die verfügbare Query-Syntax ist zur Suche im Index nutzbar.

Beim Einsatz von Lucene können sich allerdings eventuell noch einige Schwierigkeiten ergeben. Wer bereits die eine oder andere Anwendung auf Basis von Lucene geschrieben hat, weiß, dass derselbe Code oft immer wieder geschrieben werden muss. Außerdem bietet Lucene von Haus aus keinen Mechanismus an, um mit sehr großen Datenmengen umzugehen oder einen Index aus Gründen der Lastverteilung oder Ausfallsicherheit redundant auszulegen. Schließlich gibt es oft auch noch den Wunsch, die Funktionalität von Lucene in einer nicht auf der JVM laufenden Sprache zu verwenden. Für alle diese Probleme besteht die Lösung darin, einen auf Lucene aufsetzenden Suchserver einzusetzen.

Dieser Artikel stellt die beiden Open-Source-Lösungen Apache Solr und ElasticSearch vor. Als Beispiel-Anwendung dient uns wieder die Indizierung von Vortragsbeschreibungen mit ihren Meta-Informationen wie „Sprecher“ oder „Datum“. Im ersten Teil werfen wir einen Blick auf den Platzhirsch Apache Solr, bevor wir uns danach anschauen, was der Newcomer ElasticSearch anders macht.

Apache Solr

Im Gegensatz zu Lucene ist Solr keine Bibliothek, sondern eine Anwendung, die die Funktionalität von Lucene per HTTP zur Verfügung stellt. Solr kann entweder als Web Application Archive (WAR) in einen Servlet-Container eingesetzt oder direkt per „embedded Jetty“ gestartet werden. Die eigene Anwendung greift dann zum Indizieren und Suchen per HTTP auf Solr zu (siehe Abbildung 1). Zum Austausch können verschiedene Formate wie XML, JSON, CSV oder bei einer Java-Anwendung ein spezielles binäres Format zum Einsatz kommen. Es existieren zahlreiche Client-Implementierungen für unterschiedliche Programmiersprachen; für Java wird SolrJ direkt vom Solr-Team weiterentwickelt.

Das Schema der in Solr indizierten Daten muss vor der Verwendung in der Konfigurationsdatei „schema.xml“ hinterlegt sein. Dabei werden den Feldern der Dokumente Typen zugewiesen, wobei sowohl primitive Typen als auch selbst konfigurierte möglich sind. Für eigene Typen lässt sich eine Analyzer-Kette für das Indizieren und die Suche hinterlegen. Bei jedem Zugriff auf dieses Feld wird dann automatisch der konfigurierte Analyzer verwendet.

Listing 1 zeigt einige Felder unserer Beispiel-Dokumente und die Konfiguration des Typs für deutschen Text. Wie von Lucene bekannt, werden am Feld die Attribute „indexed“ und „stored“ gesetzt, um zu bestimmen, ob die Analyzer-Kette durchlaufen werden muss und ob die Originalwerte bei der Ausgabe der Suchergebnisse verfügbar sein sollen.

Nach Konfiguration unseres Schemas und einem Neustart können wir, wie in Listing 2 angegeben, Daten indizieren. Der Zugriff auf Solr über SolrJ erfolgt über Implementierungen des Interface „SolrServer“, das alle notwendigen Methoden zur Verfügung stellt. Vergleichbar mit der Lucene-Implementierung werden Document-Instanzen erzeugt, denen unsere Felder hinzugefügt werden. Es sind jedoch jetzt weniger Informationen anzugeben. Ob ein Feld „analyzed“ ist oder gespeichert wird, ist nun direkt in der Solr-Konfiguration hinterlegt. Auch müssen wir an keiner Stelle im Client den Analyzer angeben, dieser wird durch den Feldtyp automatisch bestimmt. Dadurch sind für Anpassungen an der Suchlogik oft nur Änderungen an der Konfiguration in Solr notwendig.

Neben dem Schema sind für die Suche noch weitere anwendungsspezifische Konfigurationen notwendig. In „solrconfig.xml“ können neben diversen Settings wie Lucene-Einstellungen oder Caching-Optionen auch die zentralen „RequestHand-

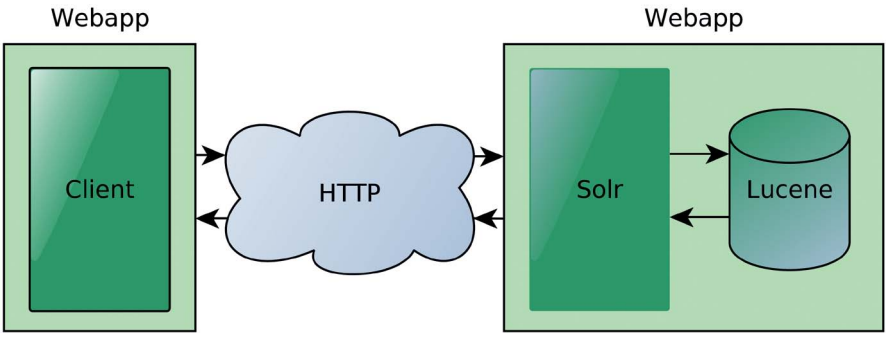


Abbildung 1: Kommunikation mit Solr

```
<fields>
  <field name="title" type="text_de" indexed="true" stored="true"/>
  <field name="category" type="string" indexed="true" stored="true" multiValued="true"
omitNorms="true"/>
  <field name="date" type="date" indexed="true" stored="true"/>
  <field name="speaker" type="string" indexed="true" stored="true" multiValued="true"/>
  <field name="content" type="text_de" indexed="true" stored="true"/>
</fields>
<types>
  <fieldType name="text_de" class="solr.TextField" positionIncrementGap="100">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.GermanNormalizationFilterFactory"/>
        <filter class="solr.GermanLightStemFilterFactory"/>
      </analyzer>
  </fieldType>
  [...]
</types>
```

Listing 1: Solr-Feldkonfiguration in schema.xml

ler" und „SearchComponents" konfiguriert werden. SearchComponents bieten jeweils unterschiedliche Aspekte einer Suche an, beispielsweise „QueryComponent" für die eigentliche Kernfunktionalität der Suche oder „HighlighterComponent" zum Hervorheben von Treffern im Text.

Die eigentliche Funktionalität nach außen stellen die „RequestHandler" bereit, die die „SearchComponents" verwenden und einen HTTP-Endpunkt zur Verfügung stellen. Häufig verwendete „SearchComponents" sind bei Verwendung des für das Suchen vorgesehenen „SearchHandler" schon registriert. Die von den „SearchComponents" erwarteten Parameter lassen sich durch Default-Werte vordefinieren. Je nach Konfiguration und Bedarf können diese Werte aber auch durch übergebene Parameter beim Aufruf durch den Client überschrieben werden (siehe Listing 3).

Oftmals genügen die von Solr mitgelieferten Komponenten völlig aus, bei Bedarf lassen sich jedoch auch eigene Komponenten einbinden. Nicht nur an dieser Stelle ist Solr leicht erweiterbar; so gut wie alle enthaltenen Komponenten sind durch eigene Implementierungen ersetzbar, ohne Anpassungen im Kern vornehmen zu müssen.

Da der Zugriff über HTTP erfolgt, kann man mit einfachen Tools die korrekte Funktionsweise testen. Der in Listing 4 angegebene Aufruf mit dem Kommandozeilen-Werkzeug „curl" fordert beispielsweise die Ergebnisse zu dem Suchbegriff „lucene" auf dem „RequestHandler" namens „jug" im JSON-Format an, wobei der „row"-Parameter, der die Anzahl der Suchergebnisse bestimmt, mit einem abweichenden Wert überschrieben wird.

Durch die Nutzung von Standard-Protokollen lassen sich Probleme auch auf Produktiv-Systemen leicht identifizieren. Für tiefergehende Analysen bietet Solr eine Administrations-Oberfläche an, in der unter anderem Statistiken über die Daten im Index abgerufen werden können und der Analyzing-Prozess mit Beispieldaten durchgeführt wird (siehe Abbildung 2).

Aus einer Java-Anwendung wird, wie in Listing 5 dargestellt, gesucht. Auch hier sind wieder deutlich weniger Informationen als in der Lucene-Implementierung notwendig. Der Client muss lediglich den Suchstring und den zu verwendenden „RequestHandler" übergeben. Die Konfiguration des Analyzing ist komplett von der Anwendung in Solr gewandert.

Mit wenigen Zeilen Code und Konfiguration haben wir bereits die wichtigsten Grundlagen für eine Suche geschaffen. Solr kümmert sich um die Interna des Zugriffs auf Lucene. Auch wenn uns dies bereits viel Arbeit im Vergleich zu der direkten Nutzung von Lucene abnimmt, gibt es noch weitere Vorteile. So lassen sich fortschrittliche Such-Features fast rein konfi-

```
SolrServer server = new HttpSolrServer("http://localhost:8080");

SolrInputDocument document = new SolrInputDocument();
document.addField("title", "Suchen und Finden mit Lucene und Solr");
document.addField("speaker", "Florian Hopf");
document.addField("date", "2012-07-03T22:00:00Z");

server.add(document);
server.commit();
```

Listing 2: Indizieren von Daten über SolrJ

```
<requestHandler name="/jug" class="solr.SearchHandler">
  <lst name="defaults">
    <int name="rows">10</int>
    <str name="q.op">AND</str>
    <str name="q.alt">*:*</str>
    <str name="defType">edismax</str>
    <str name="qf">content title</str>
  </lst>
</requestHandler>
```

Listing 3: Konfiguration eines „RequestHandler" in „solrconfig.xml"

gurativ hinzufügen. Einer der Faktoren für den enormen Erfolg von Solr ist beispielsweise die „Facettierung", die in Solr früher als in Lucene verfügbar war. Dabei können Suchergebnisse dynamisch zu Kategorien zusammengefasst werden, die der Nutzer zur Eingrenzung der Suchergebnisse nehmen kann. Besonders bei großen Datenmengen ist damit das Auffinden von Inhalten enorm erleichtert, da man durch die Suchergebnisse geführt wird.

Listing 6 zeigt, wie die Facettierung über Java-Code angefordert und ausgelesen werden kann. Meist sind die Ergebnisse der Facettierung als Links neben der Suchergebnisliste angezeigt. Durch einen Klick werden die Treffer dann automatisch anhand einer übergebenen Filter-Query eingeschränkt und damit die Anzahl nur auf Dokumente mit dieser Eigenschaft reduziert.

Obwohl schon früh eine Skalierung von Solr über Master-Slave-Setups möglich war, wurde dieser Aspekt mit Solr 4 neu angegangen. Seit diesem Release ist es möglich, echte Solr-Cluster zu betreiben, die ausfallsicher in Bezug auf das Indizieren und die Suche sind. Zur Verwaltung des Cluster-States wird das aus dem Hadoop-Framework stammende Apache ZooKeeper verwendet.

Wie wir gesehen haben, ist es mithilfe von Solr auch ohne tiefergehende Lucene-Kenntnisse möglich, fortschrittliche Such-Anwendungen zu bauen. Die mitgelieferten Komponenten bieten neben dem vorgestellten Faceting sehr viele Möglichkeiten, um die Suche mit weiteren Funktionalitäten anzureichern. Häufig verwendet werden neben dem schon erwähnten Highlighting beispielsweise „MoreLikeThisComponent" zur Suche nach ähnlichen Dokumenten, „Suggester" für Autovervollständigungen von Termen aus dem Index oder externen Quellen und „SpellcheckComponent" zur Korrektur von Rechtschreibfehlern im Suchbegriff. Ebenfalls erwähnenswert ist die Indizierung von Geo-Koordinaten über Längen- und Breitengrade, die entweder zur Sortierung (sortiere nach Ergebnis in der Nähe) oder zur Filterung (zeige nur Ergebnisse im Umkreis von 100 km) verwendet werden kann.

ElasticSearch

Das seit 2010 ebenfalls als Open Source verfügbare ElasticSearch ähnelt Solr auf den ersten Blick. Beides sind fertige Anwendungen, die Lucene benutzen und die Funktionalität über eine HTTP-Schicht abstrahieren. Beide bieten zudem die Möglichkeit, auch fortgeschrittene Suchfunktionalität direkt zu nutzen.

Im Detail macht ElasticSearch jedoch einiges anders. Im Gegensatz zu Solr ist es keine auf JEE basierende Web-, sondern eine Stand-alone-Anwendung. Die Netzwerk-Funktionalität ist über das asynchrone Netzwerk-Framework „Netty" im-

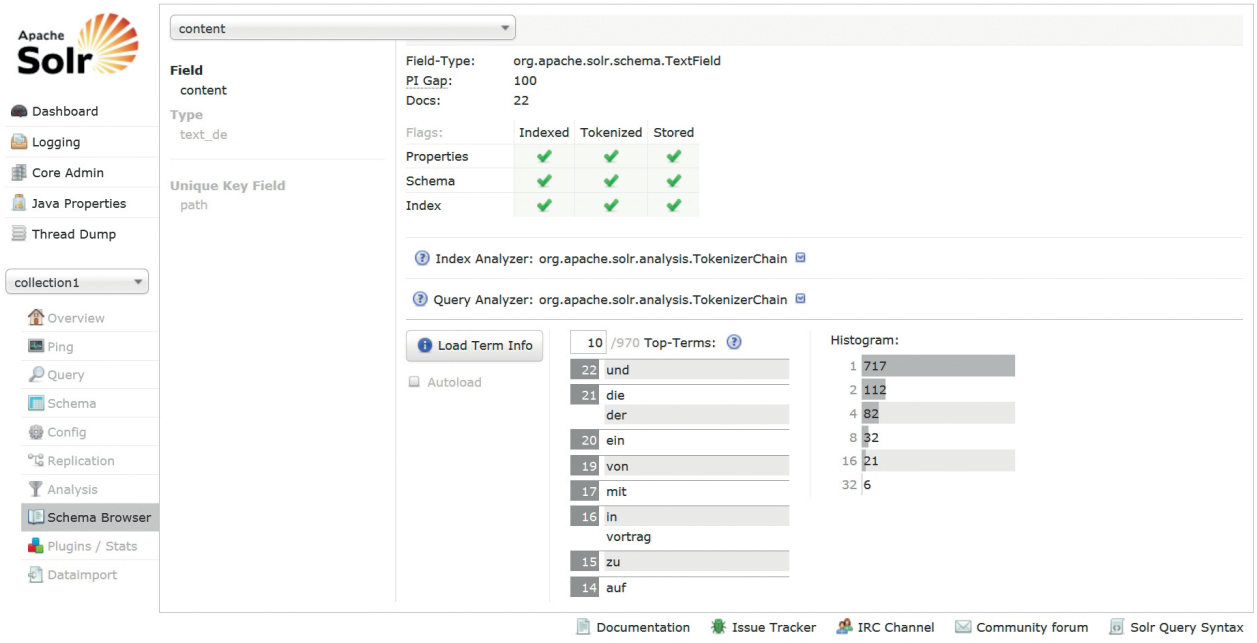


Abbildung 2: Solr-Administrations-Oberfläche

```
curl “http://localhost:8080/solr/jug?q=lucene&rows=1&wt=json
&indent=on“
{
  “responseHeader“:{
    “status“:0,
    “QTime“:0},
  “response“:{“numFound“:3,“start“:0,“docs“:[
      {
        “title”:”Suchen und Finden mit Lucene und Solr”,
        “date“:“2012-07-03T22:00:00Z“,
        “speaker“:[“Florian Hopf“],
      [...]}]
  }
}
```

Listing 4: Ausführen einer Suche über „curl“ in Solr

```
SolrQuery solrQuery = new SolrQuery(“suche“);
solrQuery.setQueryType(“/jug“);

QueryResponse response = server.query(solrQuery);
assertEquals(1, response.getResults().size());

SolrDocument result = response.getResults().get(0);
assertEquals(“Suchen und Finden mit Lucene und Solr”, re-
sult.get(“title”));
assertEquals(“Florian Hopf“, result.
getFirstValue(“speaker“));
```

Listing 5: Ausführen einer Suche mit SolrJ

```
SolrQuery query = ...;
solrQuery.setFacet(true);
solrQuery.addFacetField(“speaker“);

QueryResponse response = server.query(solrQuery);

List<FacetField.Count> speakerFacet = response.
getFacetField(“speaker“).getValues();
assertEquals(1, speakerFacet.get(0).getCount());
assertEquals(“Florian Hopf“, speakerFacet.get(0).getName());
```

Listing 6: Facettierung über SolrJ

```
curl -XPOST ‚http://localhost:9200/jug/talk/‘ -d ‚{
  “speaker” : “Florian Hopf”,
  “date” : “2012-07-04T19:15:00”,
  “title” : “Suchen und Finden mit Lucene und Solr”
}‘
{“ok“:true,“_index“:“jug“,“_type“:“talk“,
“_id”:”CeltdivQRGSvLY_dBZv1jw”,”_version”:1}
```

Listing 7: Indizierung von Daten in ElasticSearch

plementiert. Zur Kommunikation kommt ein echtes REST-API zum Einsatz, als Austausch- und internes Format dient JSON. In Bezug auf den Datenaustausch gibt es also nicht die Vielfalt an Möglichkeiten wie bei Solr. ElasticSearch benötigt allerdings keine Konfiguration vorab, nach dem Download ist es sofort einsatzbereit. Der Einstieg fällt dadurch sehr leicht. Beispielsweise lassen sich mit dem in Listing 7 angegebenen Curl-Kommando Daten ohne jegliche Konfiguration direkt nach dem Download indizieren.

ElasticSearch teilt die indizierten Daten grundsätzlich in Indizes und Typen auf. In unserem Beispiel werden der Index „jug“ und der Typ „talk“ verwendet. Existiert der Index noch nicht, wird dieser automatisch angelegt. Die zu indizierenden Daten werden über eine JSON-Struktur im Request-Body übergeben.

Der Typ bestimmt das Schema der Dokumente. Erfolgt vorab keine Konfiguration, wird das Schema aus dem ersten indizierten Dokument abgeleitet. Für unseren Fall werden für den Talk-Titel der Default-Analyzer verwendet und das Datum anhand des Formats erkannt und speziell abgelegt. Das Schema eines Typs ist über den „_mapping“-Endpunkt einsehbar, der an die URL angehängt werden kann.

Da das Analyzing oft eine anwendungsspezifische Konfiguration benötigt, kann man sich mit diesem automatisierten Prozess oft nicht zufriedengeben und will stattdessen selbst Anpassungen vornehmen. Eine nachträgliche Änderung eines einmal angelegten Schemas ist nur eingeschränkt möglich. Neue Felder können problemlos hinzugefügt, das Verhalten vorhandener Felder kann allerdings nicht mehr verändert werden. Deshalb gilt auch für ElasticSearch, dass man sich vorab Gedanken über die Ablage seiner Daten machen sollte.

In Listing 8 wird unser zuvor erstellter Index erst gelöscht, bevor er mit einem zu unseren Daten passenden Mapping neu erstellt wird. Dadurch wird das Feld „title“ mit dem von uns benötigten Analyzer für deutsche Texte verarbeitet.

Um auf den Daten zu suchen, ist das in Listing 9 angegebene „curl“-Kommando möglich. Der „_search-Handler“ akzeptiert den Query-Parameter „q“, dem ein Wert in der Lucene-Query-Syntax über-

geben werden kann. Ist kein Feldname angegeben, wird auf einem speziellen Feld „_all“ gesucht, das per Default alle Feldinhalte des Dokuments enthält. Für die spätere Ausgabe speichert ElasticSearch standardmäßig die kompletten indizierten Daten im Feld „_source“. Deshalb muss man sich nicht sofort überlegen, welche Felder man als gespeichert ablegen will; der Originalinhalt ist immer verfügbar. Bei großen Datenmengen sollte jedoch darüber nachgedacht werden, dieses Feature zu deaktivieren, um Speicherplatz zu sparen.

Wie schon bei Solr gesehen, wollen wir eventuell noch weitere Features wie die Facettierung aktivieren oder auch komplexere Queries formulieren. In ElasticSearch arbeitet man deshalb nur selten mit Query-Parametern, meist wird stattdessen die sogenannte „Query-DSL“ verwendet. Dabei handelt es sich um JSON-Strukturen, die einzelne Aspekte einer Abfrage abbilden. Listing 10 zeigt beispielsweise eine Abfrage, bei der zusätzlich die Facettierung auf ein Feld angefordert wird.

Die Verwendung einer ausdrucksstarken Sprache mag an dieser Stelle wie eine unnötige Aufblähung des Request wirken. In der Praxis ist es allerdings so, dass dies zu deutlich leichter verständlichen Beispielen führt und die Wartbarkeit einer Anwendung verbessern kann.

Wenn der Client in Java implementiert ist, muss man sich nicht zwingend selbst mit JSON auseinandersetzen. Der integrierte Java-Client stellt über Builder-Objekte eine der Query-DSL ähnliche Sicht auf die Abfrage dar. Listing 11 zeigt das vorher in „curl“ formulierte Beispiel in Java, wobei einige Methoden statisch importiert wurden. Interessanterweise spricht der Java-

```
curl -XDELETE http://localhost:9200/jug/
curl -XPOST localhost:9200/jug -d ,{
  “mappings“ : {
    “talk“ : {
      “properties“ : {
        “title“ : {
          “type“ : “string“,
          “analyzer” : “german”
        }
      }
    }
  }
}’
```

Listing 8: Löschen und Erstellen eines ElasticSearch-Index mit Mapping-Informationen

```
curl -XGET ,http://localhost:9200/jug/talk/_
search?q=title:suche‘
{...},
“hits“:{
  “total“:1,“max_score“:0.054244425,
  “hits“:[{
    ...,
    “_score“:0.054244425,
    “_source“ : {
      “speaker“ : “Florian Hopf“,
      “date” : “2012-07-04T19:15:00”,
      “title”: “Suchen und Finden mit Lucene und Solr”
    }
  }
}
```

Listing 9: Suchen in ElasticSearch per „curl”

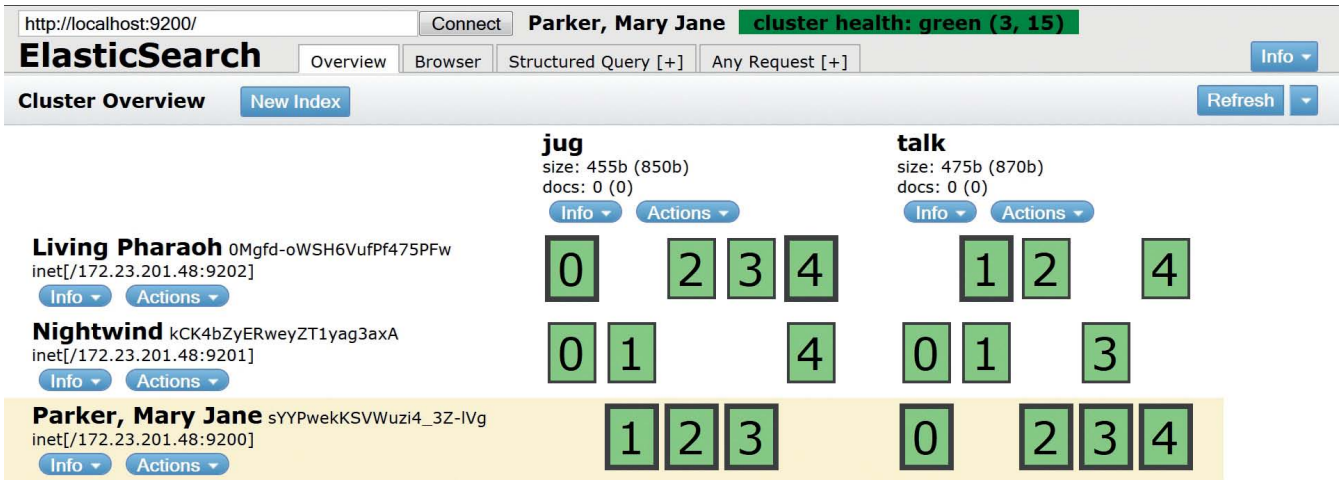


Abbildung 3: Verteilung eines Clusters, visualisiert in „elasticsearch-head“

```
curl -XGET ‚http://localhost:9200/jug/talk/_search‘ -d ‚{
  “query“ : {
    “query_string“ : {
      “query“ : “suche“
    }
  },
  “facets“ : {
    “tags“ : {
      “terms“ : {
        “field“ : “speaker“
      }
    }
  }
}‘
```

Listing 10: Suchen in ElasticSearch per Query-DSL

Client in der Default-Konfiguration auch nicht per REST-API mit dem ElasticSearch-Server, sondern hängt sich als Knoten in den Cluster ein.

Generell ist das Thema „Clustering und Verteilung in ElasticSearch“ für den Benutzer sehr einfach zu verwalten. Knoten im selben Netzwerk verbinden sich anhand eines konfigurierten Namens mit allen Knoten desselben Cluster-Namens. Für die Verteilung relevant sind „Shards“ und „Replicas“, die für jeden logischen Index konfiguriert werden können. „Shard“ ist ein Lucene-Index, in den ein Teil der Indexdaten abgelegt wird. „Replica“ ist die Kopie eines Shard, die auf einem weiteren Knoten abgelegt wird. Wie wichtig der Verteilungsgedanke bei ElasticSearch ist, sieht man auch an den vorkonfigurierten Werten: Wenn nicht anders angegeben, wird jeder logische Index auf fünf Shards verteilt. Somit ist selbst die Nutzung von nur einem Knoten mit einem logischen Index schon eine verteilte Suche auf Lucene-Ebene.

Shards und Replicas werden je nach vorhandenen Knoten automatisch von ElasticSearch verteilt. Tritt ein neuer Knoten dem Cluster bei, werden die Shards umorganisiert, damit eine möglichst gute Lastverteilung erreicht wird. Der Benutzer kann jedoch alle diese Mechanismen selbst beeinflussen; beispielsweise kann konfiguriert werden, dass bestimmte Daten nur auf festgelegten Knoten abgelegt sind. Abbildung 3 zeigt einen Screenshot des ElasticSearch-Plug-ins „elasticsearch-head“, mit dem unter anderem der aktuelle Zustand eines Clusters visualisiert werden kann. Unsere fünf Shards sind mit jeweils einem Replica auf drei Knoten verteilt.

Die Qual der Wahl

Beide Produkte eignen sich gut für die meisten Anwendungsfälle und stellen eine zukunftssichere Wahl dar. Änderungen in Lucene fließen in beide Projekte schnell ein; in Solr, da es als Unterprojekt von Lucene von denselben Committern bearbeitet wird, und in ElasticSearch, weil die dahinterstehende Firma mehrere Lucene-Committer beschäftigt. Die Projekte werden sich in nächster Zeit weiter annähern, was den Umfang der Features betrifft. Solr hat bereits Ideen von ElasticSearch übernommen und ElasticSearch hat bezüglich der Such-Features aufgeholt. Noch ist es so, dass zu Solr mehr Wissen in Form von Büchern oder Erfahrungen in Unternehmen verfügbar ist. Der schnelle Einstieg in ElasticSearch kann diesen Vorteil allerdings wieder relativieren.

Links

http://lucene.apache.org/solr
http://elasticsearch.org
https://github.com/fhopf/lucene-solr-talk

Florian Hopf
mail@florian-hopf.de

Florian Hopf arbeitet als freiberuflicher Software-Entwickler mit den Schwerpunkten „Content Management“ und „Suchlösungen“ in Karlsruhe. Er setzt Lucene und Solr seit Jahren in unterschiedlichen Projekten ein und ist einer der Organisatoren der Java User Group Karlsruhe.

```
Client esClient = nodeBuilder().client(true).node().client();
QueryBuilder query = queryString(“suche“);
TermsFacetBuilder facet = termsFacet(“speaker“).field(“speaker“);
SearchResponse response = esClient.prepareSearch(“jug“)
  .addFacet(facet)
  .setQuery(query)
  .execute().actionGet();
assertEquals(1, response.getHits().getTotalHits());
```

Listing 11: Suchen mit dem ElasticSearch-Java-Client

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG *News* und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.