Open search platform
Agenda

• What is Solr?
• Installing Solr and run it locally
• Building index
• Searching the index
What is Solr
• Web service layer built on top of Lucene library.
  – Independent programming APIs

• Provides scalability, deployed on top of application server like Tomcat.
  – Several caches for faster search responses.
Solr Architecture

Tow modes:

• Standalone:
  – Single Machine, Multi core.

• SolrCloud:
  – Multiple Machines (instances), Multible Collection (distributed).

Source: Solr In Action
Solr features

• Restful APIs.
• Spelling Correction
• Expand queries based on configurable definition list
• Auto Suggestions.
Who needs Solr?

• Programmers to develop sophisticated, high-performance search applications.

• Websites build their search service using Solr:
  – Netflix uses Solr for their site search feature.
  – The Guardian uses Solr to power it's Open Platform.
  – SourceForge uses Solr to provide faceted search across all its projects.
Tutorial Roadmap

- The tutorial is organized into three sections that each build on the one before it.
  - 1- Start Solr server.
  - 2- Create a core and index some documents.
  - 3- Perform some searches.
Installing solr

• Installation of Solr on Unix-compatible or Windows servers by extracting the package.
  – Solr System Requirements.

• Start the server:
  – Bin / slor start –p 8983
  – http://localhost:8983/solr/#/
Check the Solr status

• Command:
  – $Solr status

• If you need convincing, use a Web browser
  – localhost:8983/solr/#/
Overview of the Solr Admin UI
Working With Solr
How to interact with solr

• Solr is a Web application, but because it is built on open protocols, any type of client application can use Solr
  – Client APIs:
    Pysolr: It provides an interface that queries the server and returns results based on the query.
  – POSTing/Curl.
List of Client APIs

<table>
<thead>
<tr>
<th>Name</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SolRuby</td>
<td>Ruby</td>
</tr>
<tr>
<td>DelSolr</td>
<td>Ruby</td>
</tr>
<tr>
<td>acts_as_solr</td>
<td>Rails</td>
</tr>
<tr>
<td>Flare</td>
<td>Rails</td>
</tr>
<tr>
<td>SolPHP</td>
<td>PHP</td>
</tr>
<tr>
<td>SolrJ</td>
<td>Java</td>
</tr>
<tr>
<td>Python API</td>
<td>Python</td>
</tr>
<tr>
<td>PySolr</td>
<td>Python</td>
</tr>
<tr>
<td>SolPerl</td>
<td>Perl</td>
</tr>
<tr>
<td>Solr.pm</td>
<td>Perl</td>
</tr>
<tr>
<td>SolrForrest</td>
<td>Forrest/Cocoon</td>
</tr>
<tr>
<td>SolrSharp</td>
<td>C#</td>
</tr>
</tbody>
</table>

- The Solr Wiki contains a list of client APIs at [http://wiki.apache.org/solr/IntegratingSolr](http://wiki.apache.org/solr/IntegratingSolr)
Preparing Solr

Step A
- Create a core.
- Define schema.
- Add documents.
- Update documents.

Step B
- Query Syntax and Parsing: (Proximity search, Boosting a term, Boolean search)
- Query expansion: (synonyms graph filter)
Indexing in Solr

Example tweet in Solr’s XML format:

1. `<add>`
2. `<doc>`
3. `<field name="id">` ...
4. `<field name="screen_name">` ...
5. `<field name="type">` ...
6. `<field name="timestamp">` ...
7. `<field name="lang">` ...
8. `<field name="text">` ...
9. `</doc>`
10. `</add>`

Solr Core:

1. POST / HTTP
2. Document update service

Text Analysis:

1. Tokenizer
2. Token Filter
3. Token Filter

Terms transformed by Solr’s text analysis process stored in the Lucene index.

The contents of each field are sent through text analysis process before indexing.

Lucene Index

Example tweet:

yumm drink latte caffe grecco sf historic north beach learn text analysis #solrinaction @manning my ipad

Source: Solr In Action
A. Creating index

1-Create a core
   - In order to be able to index and search. You can do so by using: `bin/solr create -c <name>`

2-Solr’s schema
   - Provides an idea of how content is structured.
   - How field name is defined in the Schema

3- Add documents
   Populate the contents of the index.

4- Updating documents
   Update the content of the index file with new documents.
A.1 Creating Core

• contains index and configuration.
• A single server can support multiple cores and it is used for data partitioning.
• Local Documents (single node) (standalone mode).
• Cloud Collection (cluster of nodes) (SolrCloud mode).
A.2 Schema

• Schema Document Design:
• Each document consists of a list of fields.
• One field must uniquely identify each document in the index.

Which fields will your users want to search on?

What fields should be displayed in your search results?
Example schema.xml

```xml
<schema name="example" version="1.5">
  <fields>
    <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false"/>
    <field name="title" type="text_general" indexed="true" stored="true" multiValued="true"/>
    <field name="subject" type="text_general" indexed="true" stored="true"/>
    <field name="description" type="text_general" indexed="true" stored="true"/>
    <field name="comments" type="text_general" indexed="true" stored="true"/>
    <field name="author" type="text_general" indexed="true" stored="true"/>
    <field name="category" type="text_general" indexed="true" stored="true"/>
    <field name="last_modified" type="date" indexed="true" stored="true"/>
    <field name="links" type="string" indexed="true" stored="true" multiValued="true"/>
    <field name="content" type="text_general" indexed="false" stored="true" multiValued="true"/>
    <field name="text" type="text_general" indexed="true" stored="false" multiValued="true"/>
    <field name="weight" type="double" indexed="true" stored="true"/>
    <field name="price" type="float" indexed="true" stored="true"/>
    <field name="popularity" type="int" indexed="true" stored="true"/>
    <field name="inStock" type="boolean" indexed="true" stored="true"/>
    <field name="store" type="location" indexed="true" stored="true"/>
    <dynamicField name="*_s" type="string" indexed="true" stored="true"/>
    <dynamicField name="*_dt" type="date" indexed="true" stored="true"/>
  </fields>
  <uniqueKey>id</uniqueKey>
  <copyField source="title" dest="text"/>
  <copyField source="author" dest="text"/>
  <copyField source="description" dest="text"/>
  <copyField source="keywords" dest="text"/>
  <copyField source="content" dest="text"/>
</schema>
```

schema (a file named either managed-schema or schema.xml)
<types>
  <fieldType name="string" class="solr.StrField" sortMissingLast="true" />
  <fieldType name="boolean" class="solr.BoolField" sortMissingLast="true" />
  <fieldType name="int" class="solr.TrieIntField" precisionStep="0" positionIncrementGap="0"/>
  <fieldType name="float" class="solr.TrieFloatField" precisionStep="0" positionIncrementGap="0"/>
  <fieldType name="long" class="solr.TrieLongField" precisionStep="0" positionIncrementGap="0"/>
  <fieldType name="double" class="solr.TrieDoubleField" precisionStep="0" positionIncrementGap="0"/>
  <fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.LowerCaseFilterFactory"/>
    <analyzer type="index">
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
      <filter class="solr.LowerCaseFilterFactory"/>
      <analyzer type="query">
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
        <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
        <filter class="solr.LowerCaseFilterFactory"/>
      </analyzer>
    </analyzer>
  </fieldType>
</types>
</schema>
Retrieval models

• One of the available retrieval model is Best Matching (BM25)

1-Make the following changes to the schema.xml file:

```xml
<similarity class="solr.BM25SimilarityFactory">
    <float name="k1">1.3</float>
    <float name="b">0.76</float>
</similarity>
```

2-Delete and re-index all the documents: (you can use the browser):

Retrieval models

3- Re-indexing the files run the following commands on the console:

- (windows users) `java Dc=corename -jar post.jar *.xml`
- (linux/unix) `post corename *.xml`
A.3 Add documents

- The three most common ways of loading data into a Solr index:
  - Using the Solr Cell framework built on Apache Tika (support multiformat).
  - Uploading XML files by sending HTTP requests to the Solr server from any environment where such requests can be generated.
  - Writing a custom application to ingest data through Solr’s Client API.
A.4 Updating document

• Solr supports three approaches to updating documents that have only partially changed:
  - **Atomic updates:** This approach allows changing only one or more fields of a document without having to re-index the entire document.
  - **In-place updates:** This approach is similar to atomic updates (is a subset of atomic updates in some sense), but can be used only for updating single valued non-indexed and non-stored docValue-based numeric fields.
  - **Optimistic:** concurrency or optimistic locking. It is a feature of many NoSQL databases, and allows conditional updating a document based on its version.

Atomic updates, the entire document is re-indexed. In Place, only the fields to be updated are affected and the rest of the documents are not re-indexed internally.
Index update

Index update commands can be sent as XML message to the update handler using

```
  <add>  
    <doc>  
      <field name="authors">Patrick Eagar</field>  
      <field name="subject">Sports</field>  
      <field name="dd">796.35</field>  
      <field name="isbn">0002166313</field>  
      <field name="yearpub">1982</field>  
      <field name="publisher">Collins</field>  
    </doc>  
  </add>'
```

- posting XML messages contained in a file, you can use the alternative form:
In place update

```json
{
  "id": "mydoc",
  "price": 10,
  "popularity": 42,
  "categories": ["kids"],
  "promo_ids": ["a123x"],
  "tags": ["free_to_try", "buy_now", "clearance", "on_sale"]
}
```

Atomic-update

```json
{
  "id": "mydoc",
  "price": { "set": 99 },
  "price": { "set": 99 },
  "popularity": { "inc": 20 },
  "categories": { "add": ["toys", "games"] },
  "promo_ids": { "remove": "a123x" },
  "tags": { "remove": ["free_to_try", "on_sale"] }
}
```

```json
{
  "id": "mydoc",
  "price": 99,
  "popularity": 62,
  "categories": ["kids"],
  "promo_ids": ["a123x"],
  "tags": ["free_to_try", "buy_now", "clearance", "on_sale"]
}
```
A.4 Updating document (optimistic)

Example:

1- Edit any of the existing example data files, change some of the data,

2- Re-run the PostTool (bin/post).

You’ll see your changes reflected in subsequent searches
B. Search

Solr’s default Query Parser is also known as the “lucene” parser
# The Standard Query Parser

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>Defines a query using standard query syntax. This parameter is mandatory.</td>
</tr>
<tr>
<td>q.op</td>
<td>Specifies the default operator for query expressions, overriding the default operator specified in the Schema. Possible values are &quot;AND&quot; or &quot;OR&quot;.</td>
</tr>
<tr>
<td>df</td>
<td>Specifies a default field, overriding the definition of a default field in the Schema.</td>
</tr>
<tr>
<td>sow</td>
<td>Split on whitespace: if set to <code>false</code>, whitespace-separated term sequences will be provided to text analysis in one shot, enabling proper function of analysis filters that operate over term sequences, e.g. multi-word synonyms and shingles. Defaults to <code>true</code>: text analysis is invoked separately for each individual whitespace-separated term.</td>
</tr>
</tbody>
</table>
Sample Query

http://localhost:8983/solr/techproducts/select?q=id:SP2514N

- Multiple alternatives:
  - Proximity Searches:
    “Car speed”~5
  - Boolean Search:
    Car AND speed
  - Boosting a term with ^:
    Data^4 Control
- Specifying Fields:
  title: "Information" AND text:data
Query Expansion

• Using synonyms is a form of query expansion.

• A query for text:TV will expand into (text:TV text:Television)
Types of Expansion

Index-Time Expansion
- Dog?
- dog/hound/pooch
- Index analyzer
- dog hound pooch

Query-Time Expansion
- Dog?
- Hound?
- Pooch?
- Query analyzer
- dog hound pooch
Query Expansion

1- specify the synonyms file:
   - A. Using predefined set of synonyms.
   - B. Define your own file, depends on SolrSynonymParser.

   - synonyms.txt file example:

   ```
   couch, sofa, divan
teh => the
huge, ginormous, humungous => large
small => tiny, teeny, weeny
   ```
Query Expansion

2- Modify the schema configuration file.

A- Using predefined set of synonyms

Example: With this configuration the set of mappings is named "english" and can be managed via /solr/collection_name/schema/analysis/synonyms/english

```xml
<analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.ManagedSynonymFilterFactory" managed="english"/>
</analyzer>
```
Query Expansion

B- Use SynonymGraph for your own list of synonyms.

```xml
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.SynonymGraphFilterFactory" synonyms="mysynonyms.txt"/>
  <filter class="solr.FlattenGraphFilterFactory"/>
</analyzer>

<analyzer type="query">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.SynonymGraphFilterFactory" synonyms="mysynonyms.txt"/>
</analyzer>
```
Query Expansion (example)

• *synonyms.txt* file:

  couch, sofa, divan
  teh => the
  huge, ginormous, humungous => large
  small => tiny, teeny, weeny

• **q**="teh ginormous, humungous sofa"

• **Result:** "the"(1), "large"(2), "large"(3), "couch"(4), "sofa"(4), "divan"(4)
Wrapup

• We’ve only scratched the surface of the available options in Solr.
  
  If you can dream it, it might be possible!
Thank you

Abeer ALDayel

https://abeeraldayel.github.io/
Solr Resources

• Apache Solr 3 Enterprise Search Server, David Smiley and Eric Pugh Packt Publishing.
• Solr In Action, Trey Grainger and Timothy Potter, Manning Publications.