CLOUDERA SOLR POC

OUTCOMES & FINDINGS

Admin

2016

**Contents**

[**2** **Overview** 1](#_Toc443518784)

[**3** **SOLR DEV Architecture** 2](#_Toc443518785)

[**4** **Environmental Setup** 3](#_Toc443518786)

[**4.1** **Runtime SOLR Configuration** 3](#_Toc443518787)

[4.1.1 Types of Field 4](#_Toc443518788)

[4.1.2 NGRAM Tokenizer 5](#_Toc443518789)

[4.1.3 SYnonyms 5](#_Toc443518790)

[4.1.4 Copy Fields 6](#_Toc443518791)

[4.1.5 Validation 6](#_Toc443518792)

[**4.2** **Create SOLR Collection** 6](#_Toc443518793)

[**4.3** **Create SOLR Core** 6](#_Toc443518794)

[**4.4** **Update SOLR Configuration** 7](#_Toc443518795)

[**5** **Indexing records from SQL Server using Data Import Handler** 7](#_Toc443518796)

[**6** **Real Time Indexing using Flume & Morphlines** 9](#_Toc443518797)

[**6.1** **Configure Flume sql-source** 10](#_Toc443518798)

[**7** **Indexing of Nested Documents (Scope Search)** 18](#_Toc443518799)

[**8** **SOLR search and output for nested documents** 20](#_Toc443518800)

[8.1 **Use Case # 1** 22](#_Toc443518801)

[**8.2** **Use Case # 2** 23](#_Toc443518802)

[**8.3** **Use Case # 3** 25](#_Toc443518803)

[**8.4** **Use Case # 4** 26](#_Toc443518804)

[**8.5** **Use Case # 5** 29](#_Toc443518805)

[**9** **Initial Indexing** 31](#_Toc443518806)

[**10** **Incremental Indexing** 32](#_Toc443518807)

[**11** **Home Page Search Queries** 32](#_Toc443518808)

[**12** **Proximity Search, Search Operators, Wild Card Character Search** 34](#_Toc443518809)

[**13** **Time Dimension Navigator** 35](#_Toc443518810)

[**14** **Performance Improvement** 36](#_Toc443518811)

[**15** **Appendix** 36](#_Toc443518812)

[15.1 Schema.xml 36](#_Toc443518813)

[15.2 Solrconfig.xml 36](#_Toc443518814)

# **Overview**

CSN application had been designed as a search centric application using FAST ESP at its core. For over past eight years the application features have grown significantly and so was the complexity. The application extensively uses some critical features of FAST ESP such as Scope Search, N-Gram, Find Similar and so on. The application have been successfully supporting the need of Law Enforcement agencies for almost a decade. However as Microsoft (current owner) of the FAST ESP has announced the End of Life for FAST ESP in 2018, we need to look for an alternate solution to replace FAST ESP. The obvious alternate solution being SOLR a POC effort has been undertaken with objective of ensuring feasibility of implementing all existing FAST ESP features through SOLR. A Cloudera SOLR instance have been selected for this exercise to be in sync with overall technology stack being proposed for CSN 3.0. There were few key activities identified for the POC with the objectives of gaining substantial understanding of optimum configurations required for achieving existing capabilities of FAST ESP installation in CSN.

Activities that were identified for this POC exercise were as follows:

|  |
| --- |
| Gain Access, Validate Solr configuration in Dev CDH |
| Define Schema for Homepage Search |
| Crawl test data from and index |
| Design Homepage Search Queries and Validate Search Results |
| Design Proximity Search, Search Operators, Wild Card Character Search |
| Design Hit Highlighting, Sort, Pagination Features |
| Design Search Within Results |
| Design Role Based Document Level Access Control |
| Design Time Dimension Navigators |
| Design Query, Data Retrieval (10 sec), and Data Format (JSON), Result Views |
| Design Name Normalization Feature with Synonym, Phonetic and Ngram Searches |
| Design Scope (Parent-Child Data) Search |
| Design Indexing Mechanism on HDFS From Dev Database (Initial/Near-Real-Time) |
| Design Grouping, Aggregative Result |
| Design of Geo-coding at Index Time |
| Team Review, Feedback |
| QC of POC Design |
| Design POC finalization, Source Code, Result, Test Results, ERB Documentation Delivery |

# **SOLR DEV Architecture**

SOLR is running on Hadoop cluster. SOLR indices are distributed among different HDFS data nodes. Cloudera 5.5 being used as a Hadoop distributor. Three SOLR nodes FTCDEVSPARK01 – 03 are being used. Zoo Keeper Service manages the SOLR instances and port on which they are running. Collections can be configured to be spread over one or more nodes. Each collection returns complete result set. In case where collection is spread over two or more active nodes, each node returns a part of the result set. This improves the query response time. Alternately a collection can have one active and one passive node for redundancy.

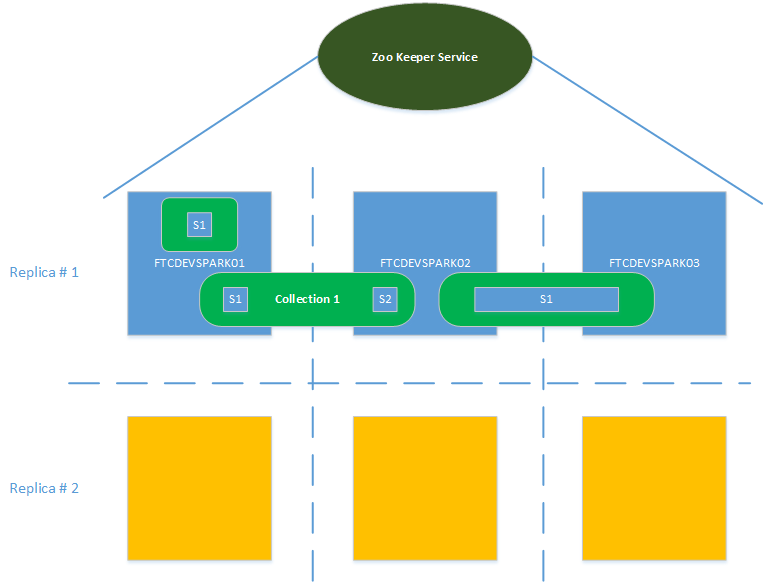


Exhibit : Logical Deployment Architecture

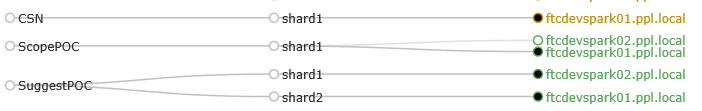


Exhibit : Representation of Physical Deployments

# **Environmental Setup**

## **Runtime SOLR Configuration**

In order to start using SOLR for indexing the data, we must configure a collection holding the index. A configuration for a collection requires a solrconfig.xml file, a schema.xml and any helper files may be referenced from the xml files. The solrconfig.xml file contains all of the SOLR settings for a given collection, and the schema.xml file specifies the schema that SOLR uses when indexing documents. Configuration files for a collection are managed as part of the instance directory. An empty instance directory can be created using following command

*$ solrctl instancedir --generate $HOME/solr\_configs*

SOLRCTL command is the preferred option with CLOUDERA SOLR while managing collections.

Then we can edit and customize it by directly editing the solrconfig.xml and schema.xml files that have been created in $HOME/solr\_configs/conf. Once we are satisfied with the configuration, we will make it available for SOLR to use by issuing the following command, which uploads the content of the entire instance directory to ZooKeeper:

*$ solrctl instancedir --create collection1 $HOME/solr\_configs*

Highlights of the schema.xml. Original schema.xml can be referred in Appendix 1.

### Types of Field

For existing CSN search capabilities we will be needing following types of fields -

* <**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="ngram" class="solr.TextField" sortMissingLast="true"**>

<**analyzer**>

<**tokenizer class="solr.NGramTokenizerFactory" minGramSize="6" maxGramSize="14"**/>

</**analyzer**>

</**fieldType**>

<fieldType name="text\_syn" class="solr.TextField" sortMissingLast="true">

<analyzer type="query">

<tokenizer class="solr.WhitespaceTokenizerFactory"/>

<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false"/>

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

<analyzer>

<tokenizer class="solr.WhitespaceTokenizerFactory" />

<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false"/>

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

</fieldType>

### NGRAM Tokenizer

NGram Tokenizers are available OOTB where we can define minimum NGram Size to Max NGram Size. Based on experience an optimum range of 6 to 14 is used for NGram Size. An example of NGram tokenizer is illustrated below –

“Bank of America” is tokenized to have following tokens (not all ) –

"Bank o", "Bank of", "Bank of ", "Bank of A", "Bank of Am", "Bank of Ame", "Bank of Amer", "Bank of Ameri"

### SYnonyms

Synonyms are captured as part of synonym configuration file (dictionary). The dictionary can support one directional conversion and expansion. For CSN Name Normalization single directional conversion is desired. The format is simple and as below

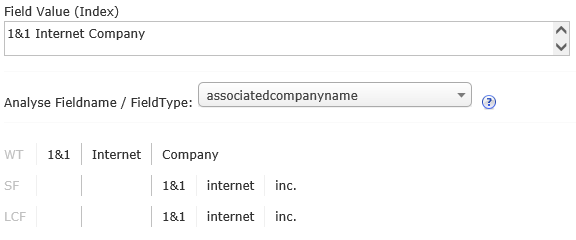
Variation => Base Form

The dictionary can be named anything of choice. There can be multiple synonyms dictionary per instance. Synonym field types are defined by specifying analyzers for query and indexing time. Optionally synonym behavior during query time and indexing time can be different. Analyzer for synonyms accepts the synonym dictionary name as an attribute.

With following entry in the synonym dictionary

1&1 Internet Company => 1&1 Internet Inc.

For “1&1 Internet Company” as the company name the entry is converted as below to its base form



### Copy Fields

As for fields like company names we need to support both synonyms match as well as NGram, same cannot be accommodated within a single file. We therefore create additional fields for NGram and copy the value of original field to this field using Copy Fields.

### Validation

We can use the solrctl tool to verify that our instance directory uploaded successfully and is available to ZooKeeper. We can use the solrctl to list the contents of an instance directory as follows:

*$ solrctl instancedir --list*

the --list command should return collection1.

## **Create SOLR Collection**

By default, the SOLR server comes up with no collections. We create our collection using the instancedir that we provided to SOLR in previous steps by using the same collection name.

*$ solrctl collection --create collection1*

We should be able to check that the collection is active. For example, for the server ftcdevspark01.ppl.local, we should be able to navigate to http://ftcdevspark01.ppl.local:8983/solr/collection1/select?q=\*%3A\*&wt=json&indent=true and verify that the collection is active. Similarly, we should also be able to observe the topology of our SolrCloud using a URL similar to: [http://](http://myhost.example.com:8983/solr/#/~cloud)ftcdevspark01.ppl.local[:8983/solr/#/~cloud](http://myhost.example.com:8983/solr/#/~cloud)

## **Create SOLR Core**

We can execute following command on a given SolrCloud node to create a core.

*$ solrctl core --create CSN*

## **Update SOLR Configuration**

We can update or modify an existing SolrCloud's copy of an instance directory based on the files present in a local filesystem. This can be thought of first using --delete collection1 followed by --create collection1 path.

*$ solrctl instancedir --reload collection1*

Then we also have to reload core using following command.

*$ solrctl core --reload CSN*

# **Indexing records from SQL Server using Data Import Handler**

The DataImportHandler is a SOLR contrib that provides a configuration driven way to import data from relational database or XML files into SOLR in both "full builds" and using incremental delta imports. With DataImportHandler we can -

* Read data residing in relational databases
* Build SOLR documents by aggregating data from multiple columns and tables according to configuration
* Update SOLR with such documents
* Provide ability to do full imports according to configuration
* Detect inserts/update deltas (changes) and do delta imports (we assume a last-modified timestamp column for this to work)
* Schedule full imports and delta imports
* Read and Index data from xml/(http/file) based on configuration
* Make it possible to plugin any kind of datasource (ftp,scp etc) and any other format of user choice (JSON,csv etc)

The data import handler has to be registered in the solrconfig.xml as follows -

*<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">*

*<lst name="defaults">*

*<str name="config">/home/pbose/solr\_configs/conf/db-data-config.xml</str>*

*</lst>*

*</requestHandler>*

The configuration is provided in two places:

* solrconfig.xml. The data config file location is added here. The datasource also can be added here or it can be put directly into the data-config.xml. Please refer the complete solrconfig.xml at Appendix 2
* data-config.xml
  + How to fetch data (queries,url etc)
  + What to read ( resultset columns, xml fields etc)
  + How to process (modify/add/remove fields)

Our data-config.xml is as follows -

<**dataConfig**>

<**dataSource driver="com.microsoft.sqlserver.jdbc.SQLServerDriver"**

**url="jdbc:sqlserver://ftcdevsql01.ppl.local:1433;database=Fuzzylogic" user="cis\_fast\_user"**

**password="\*\*\*\*"**/>

<**document**>

<**entity name="datarecord" query="**

**SELECT TOP 1000 [documentid]**

**,[documentid] as recordid**

**,[referencenumber]**

**,convert(int,referencenumber) as intreferencenumber**

**,[prodservicedesc]**

**,[consumercity]**

**,[consumerstate]**

**,[consumercountry]**

**, [primarycompanyname]**

**,[primarycompanycity]**

**,[primarycompanystate]**

**,[primarycompanyzip]**

**,[primarycompanycountry]**

**,[primarycompanyareacode]**

**,[primarycompanyphonenumber]**

**,[associatedcompanyname]**

**,[associatedcompanycity]**

**,[associatedcompanystate]**

**,[associatedcompanyzip]**

**,[associatedcompanycountry]**

**,[associatedcompanyareacode]**

**,[associatedcompanyphonenumber]**

**,[subphone]**

**,[subjectphone]**

**FROM [CISBETA].[dbo].[FAST\_CISMain\_InitialLoad]"**>

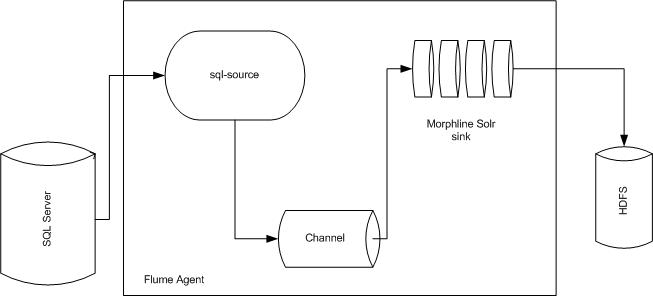
</**entity**>

</**document**>

</**dataConfig**>

# **Real Time Indexing using Flume & Morphlines**

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.



Flume has three components Source, Channel & Sink. A Flume agent is a (JVM) process that hosts the components through which events flow from an external source to the next destination (hop). A Flume source consumes events delivered to it by an external source like a SQL server. The external source sends events to Flume in a format that is recognized by the target Flume source. When a Flume source receives an event, it stores it into one or more channels. The channel is a passive store that keeps the event until it’s consumed by a Flume sink. The file channel is one example – it is backed by the local filesystem. The sink removes the event from the channel and puts it into an external repository like HDFS (via Flume HDFS sink) or forwards it to the Flume source of the next Flume agent (next hop) in the flow. The source and sink within the given agent run asynchronously with the events staged in the channel.

In our case sql-source acts as a flume source, system memory acts as a flume channel and solr-sink acts a flume sink. We need to configure our source, channel & sink in flume.conf located in /opt/cloudera/parcels/CDH-5.5.1-1.cdh5.5.1.p0.11/lib/flume-ng/conf.

## **Configure Flume sql-source**

* add flume-ng-sql-source-.jar into flume plugins dir folder

*$ mkdir -p $FLUME\_HOME/plugins.d/sql-source/lib $FLUME\_HOME/plugins.d/sql-source/libext*

*$ cp flume-ng-sql-source-0.8.jar $FLUME\_HOME/plugins.d/sql-source/lib*

* add SQL Server JDBC driver

*$ cp sqljdbc.jar $FLUME\_HOME/plugins.d/lib/sql-source/libext*

* configure flume.conf as follows

sqlagent.sources = sql-source

sqlagent.sinks = solrSink

sqlagent.channels = memoryChannel

sqlagent.sources.sql-source.type = org.keedio.flume.source.SQLSource

sqlagent.sources.sql-source.connection.url =jdbc:sqlserver://ftcdevsql01.ppl.local:1433;database=cisbeta

sqlagent.sources.sql-source.user = cis\_fast\_user

sqlagent.sources.sql-source.password = cis4ftc

sqlagent.sources.sql-source.table = dbo.FAST\_CISMain\_InitialLoad\_01192016

sqlagent.sources.sql-source.max.rows = 1000

sqlagent.sources.sql-source.columns.to.select = documentid,documentid as recordid,referencenumber,convert(int,referencenumber) as intreferencenumber,pcxml as company, conxml as consumer

sqlagent.sources.sql-source.incremental.value = 0

sqlagent.sources.sql-source.run.query.delay=360000

sqlagent.sources.sql-source.status.file.path = /var/lib/flume

sqlagent.sources.sql-source.status.file.name = sql-source.status

agent.sources.sql-source.batch.size = 1000;

sqlagent.sources.sql-source.channels = memoryChannel

sqlagent.sinks.solrSink.type = org.apache.flume.sink.solr.morphline.MorphlineSolrSink

sqlagent.sinks.solrSink.channel = memoryChannel

sqlagent.sinks.solrSink.batchSize=1000

sqlagent.sinks.solrSink.batchDurationMills=1000

sqlagent.sinks.solrSink.morphlineFile=/opt/cloudera/parcels/CDH-5.5.1-1.cdh5.5.1.p0.11/lib/flume-ng/conf/morphlines.conf

sqlagent.sinks.solrSink.morphlineId=morphline1

sqlagent.channels.memoryChannel.type = memory

sqlagent.channels.memoryChannel.capacity = 10000

sqlagent.channels.memoryChannel.transactionCapacity = 10000

* configure morphline.conf as follows

ZK\_HOST : "ftcdevcdh02.ppl.local:2181/solr"

SOLR\_LOCATOR : {

collection : ScopePOC

zkHost : ${ZK\_HOST}

}

morphlines : [

{

id : morphline1

importCommands : ["org.kitesdk.\*\*", "org.apache.solr.\*\*"]

commands : [

{

readCSV {

separator : ","

columns : [documentid,recordid,referencenumber,intreferencenumber,company,consumer]

charset : UTF-8

addEmptyStrings : false

quoteChar : "\""

}

}

{

java {

imports : """

import org.kitesdk.morphline.api.Record;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMResult;

import javax.xml.transform.stream.StreamSource;

import java.io.ByteArrayInputStream;

"""

code : """

String xmlString = (String) record.getFirstValue("company");

Record newRecord = new Record();

newRecord.put("recordid", record.getFirstValue("recordid"));

newRecord.put("referencenumber", record.getFirstValue("referencenumber"));

newRecord.put("intreferencenumber", record.getFirstValue("intreferencenumber"));

final String documentid = record.getFirstValue("documentid").toString();

newRecord.put("documentid", documentid);

newRecord.put("type\_s", "record");

newRecord.put("consumer",(String) record.getFirstValue("consumer"));

try {

DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = builderFactory.newDocumentBuilder();

Document document = builder.newDocument();

byte[] buffer = xmlString.getBytes();

ByteArrayInputStream stream = new ByteArrayInputStream(buffer);

StreamSource source = new StreamSource(stream);

DOMResult result = new DOMResult(document);

TransformerFactory tFactory = TransformerFactory.newInstance();

Transformer transformer = tFactory.newTransformer();

transformer.transform(source, result);

final Element documentElement = document.getDocumentElement();

final NodeList companyNodes = documentElement.getElementsByTagName("pc");

int size = companyNodes.getLength();

for (int i = 0; i < size; i++) {

Record childRecord = new Record();

childRecord.put("recordid", record.getFirstValue("recordid"));

childRecord.put("referencenumber", record.getFirstValue("referencenumber"));

childRecord.put("intreferencenumber", record.getFirstValue("intreferencenumber"));

childRecord.put("documentid", documentid + "C" + i);

childRecord.put("type\_s", "company");

final Element company = (Element) companyNodes.item(i);

try{childRecord.put("primarycompanyname", company.getElementsByTagName("name").item(0).getTextContent());}catch(Exception e){}

try{childRecord.put("primarycompanyareacode", company.getElementsByTagName("areacode").item(0).getTextContent());}catch(Exception e){}

try{childRecord.put("primarycompanyphonenumber", company.getElementsByTagName("phonenumber").item(0).getTextContent());}catch(Exception e){}

newRecord.put("\_loadSolr\_childDocuments", childRecord);

}

} catch (Exception e) {

e.printStackTrace();

}

return child.process(newRecord);

"""

}

}

{

java {

imports : """

import org.kitesdk.morphline.api.Record;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMResult;

import javax.xml.transform.stream.StreamSource;

import java.io.ByteArrayInputStream;

"""

code : """

String xmlString = (String) record.getFirstValue("consumer");

record.removeAll("consumer");

final String documentid = record.getFirstValue("documentid").toString();

try {

DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = builderFactory.newDocumentBuilder();

Document document = builder.newDocument();

byte[] buffer = xmlString.getBytes();

ByteArrayInputStream stream = new ByteArrayInputStream(buffer);

StreamSource source = new StreamSource(stream);

DOMResult result = new DOMResult(document);

TransformerFactory tFactory = TransformerFactory.newInstance();

Transformer transformer = tFactory.newTransformer();

transformer.transform(source, result);

final Element documentElement = document.getDocumentElement();

final NodeList consumerNodes = documentElement.getElementsByTagName("con");

int size = consumerNodes.getLength();

for (int i = 0; i < size; i++) {

Record childRecord = new Record();

childRecord.put("recordid", record.getFirstValue("recordid"));

childRecord.put("referencenumber", record.getFirstValue("referencenumber"));

childRecord.put("intreferencenumber", record.getFirstValue("intreferencenumber"));

childRecord.put("documentid", documentid + "Con" + i);

childRecord.put("type\_s", "consumer");

final Element consumer = (Element) consumerNodes.item(i);

childRecord.put("consumercity", consumer.getElementsByTagName("city").item(0).getTextContent());

childRecord.put("consumerstate", consumer.getElementsByTagName("state").item(0).getTextContent());

childRecord.put("consumercountry", consumer.getElementsByTagName("country").item(0).getTextContent());

record.put("\_loadSolr\_childDocuments", childRecord);

}

} catch (Exception e) {

e.printStackTrace();

}

return child.process(record);

"""

}

}

{

loadSolr {

solrLocator : ${SOLR\_LOCATOR}

}

}

{

java {

imports : """

import org.kitesdk.morphline.api.Record;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

"""

code : """

String documentid = (String) record.getFirstValue("documentid");

Connection conn = null;

try {

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

conn = DriverManager.getConnection("jdbc:sqlserver://ftcdevsql01.ppl.local:1433;database=cisbeta", "cis\_fast\_user", "cis4ftc");

Statement statement = conn.createStatement();

String queryString = "delete from dbo.FAST\_CISMain\_InitialLoad\_01192016 where documentid = '" + documentid + "'";

int value = statement.executeUpdate(queryString);

} catch (Exception e) {

e.printStackTrace();

} finally {

if (conn != null) try {

conn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

return child.process(record);

"""

}

}

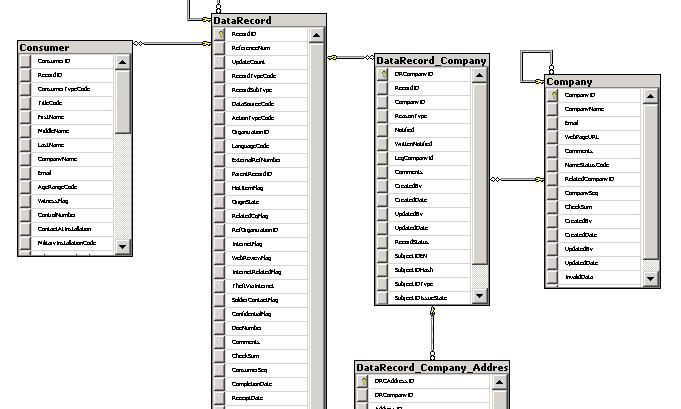
]

}

]

# **Indexing of Nested Documents (Scope Search)**

CSN records have multilevel nested relationships between entities. DataRecords can have one or more Companies. Companies can also have one or more Addresses.



For indexing in FAST ESP we did roll up chilled entities records and collate multiple items into a single multi value field. However having doing so, we lost the significance of relationships. To honor relationships FAST ESP supported Scope Search capabilities. In SOLR we achieve the same objective via use of nested documents. Nested document structure makes use of special \_childdocuments directives to indicate a child entity. For below example record with recordid 453678 has two companies.

{

**"recordid"**: **"453678"**,

**"referencenumber"**: **"34543654"**,

**"intreferencenumber"**: 34543654,

**"documentid"**: **"7653844"**,

**"type\_s"**: **"record"**,

**"\_childDocuments\_"**: [

{

**"primarycompanyname"**: **"Santanta Lizarde Corp."**,

**"city"**: **"Baltimore"**,

**"state"**: **"District of Columbia"**,

**"areacode"**: 61776

},

{

**"primarycompanyname"**: **"Lakes Kelm Ten"**,

**"city"**: **"Tulsa"**,

**"state"**: **"Minnesota"**,

**"areacode"**: 97531

}

]

}

For processing Nested Documents through Morphlines, we need minimum Kites SDK version 1.0. The recommended version is 1.1 which can be downloaded from <http://central.maven.org/maven2/org/kitesdk/kite-morphlines-solr-core/1.1.0/>

# **SOLR search and output for nested documents**

Below are few sample records with company and consumer information structured for SOLR indexing. For the sake of clarity we have highlighted the sections with different colors. Parent Record Details , Company Details , Consumer Details

|  |  |  |
| --- | --- | --- |
| **Record # 1** | **Record # 2** | **Record # 3** |
| {  "recordid": [  123  ],  "referencenumber": [  "123"  ],  "intreferencenumber": [  123  ],  "documentid": [  "123"  ],  "type\_s": [  "record"  ],  "\_childDocuments\_": [  {  "recordid": [  123  ],  "referencenumber": [  "123"  ],  "intreferencenumber": [  123  ],  "documentid": [  "123C1"  ],  "primarycompanyname": [  "ABB"  ],  "primarycompanyareacode": [  "781"  ],  "primarycompanyphonenumber": [  "2732133"  ],  "type\_s": [  "company"  ]  },  {  "recordid": [  123  ],  "referencenumber": [  "123"  ],  "intreferencenumber": [  123  ],  "documentid": [  "123C2"  ],  "primarycompanyname": [  "ATT"  ],  "primarycompanyareacode": [  "781"  ],  "primarycompanyphonenumber": [  "2732233"  ],  "type\_s": [  "company"  ]  },  {  "recordid": [  123  ],  "referencenumber": [  "123"  ],  "intreferencenumber": [  123  ],  "documentid": [  "123CM1"  ],  "consumercity": [  "Boston"  ],  "consumerstate": [  "MA"  ],  "consumercountry": [  "USA"  ],  "type\_s": [  "consumer"  ]  }  ]  } | {  "recordid": [  567  ],  "referencenumber": [  "567"  ],  "intreferencenumber": [  567  ],  "documentid": [  "567"  ],  "type\_s": [  "record"  ],  "\_childDocuments\_": [  {  "recordid": [  567  ],  "referencenumber": [  "567"  ],  "intreferencenumber": [  567  ],  "documentid": [  "567C1"  ],  "primarycompanyname": [  "ABB"  ],  "primarycompanyareacode": [  "781"  ],  "primarycompanyphonenumber": [  "2732133"  ],  "type\_s": [  "company"  ]  },  {  "recordid": [  567  ],  "referencenumber": [  "567"  ],  "intreferencenumber": [  567  ],  "documentid": [  "567C2"  ],  "primarycompanyname": [  "ATT"  ],  "primarycompanyareacode": [  "781"  ],  "primarycompanyphonenumber": [  "2732233"  ],  "type\_s": [  "company"  ]  },  {  "recordid": [  567  ],  "referencenumber": [  "567"  ],  "intreferencenumber": [  567  ],  "documentid": [  "567CM1"  ],  "consumercity": [  "Cambridge"  ],  "consumerstate": [  "MA"  ],  "consumercountry": [  "USA"  ],  "type\_s": [  "consumer"  ]  }  ]  } | {  "recordid": [  345  ],  "referencenumber": [  "345"  ],  "intreferencenumber": [  345  ],  "documentid": [  "345"  ],  "type\_s": [  "record"  ],  "\_childDocuments\_": [  {  "recordid": [  345  ],  "referencenumber": [  "345"  ],  "intreferencenumber": [  345  ],  "documentid": [  "345C1"  ],  "primarycompanyname": [  "ABB"  ],  "primarycompanyareacode": [  "409"  ],  "primarycompanyphonenumber": [  "2732133"  ],  "type\_s": [  "company"  ]  },  {  "recordid": [  345  ],  "referencenumber": [  "345"  ],  "intreferencenumber": [  345  ],  "documentid": [  "345C2"  ],  "primarycompanyname": [  "ADG"  ],  "primarycompanyareacode": [  "781"  ],  "primarycompanyphonenumber": [  "2732233"  ],  "type\_s": [  "company"  ]  }  ]  } |

## **Use Case # 1**

Trying to find records with primarycompanyareacode = 781 and primarycompanyphonenumber = 2732133 and consumercity = Boston

**Query #1**

http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=\*:\*&fq={!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company&fq={!parent which='type\_s:record'}consumercity:Boston AND type\_s:consumer

**Response # 1**

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">3</int>

<lst name="params">

<str name="q">\*:\*</str>

<arr name="fq">

<str>{!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company</str>

<str>{!parent which='type\_s:record'}consumercity:Boston AND type\_s:consumer</str>

</arr>

</lst>

</lst>

<result name="response" numFound="1" start="0">

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123</str>

<str name="type\_s">record</str>

<long name="\_version\_">1520747813059690496</long>

</doc>

</result>

</response>

## **Use Case # 2**

Trying to find records with primarycompanyareacode = 781 and primarycompanyphonenumber = 2732133 and consumerstate = MA

**Query # 2**

http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=\*:\*&fq={!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company&fq={!parent which='type\_s:record'}consumerstate:MA AND type\_s:consumer

**Response # 2**

<?xml version="1.0" encoding="UTF-8"?>

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">2</int>

<lst name="params">

<str name="q">\*:\*</str>

<arr name="fq">

<str>{!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company</str>

<str>{!parent which='type\_s:record'}consumerstate:MA AND type\_s:consumer</str>

</arr>

</lst>

</lst>

<result name="response" numFound="2" start="0">

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123</str>

<str name="type\_s">record</str>

<long name="\_version\_">1520747813059690496</long>

</doc>

<doc>

<int name="recordid">567</int>

<str name="referencenumber">567</str>

<int name="intreferencenumber">567</int>

<str name="documentid">567</str>

<str name="type\_s">record</str>

<long name="\_version\_">1520838033847353344</long>

</doc>

</result>

</response>

## **Use Case # 3**

Trying to find records with primarycompanyareacode = 781 and primarycompanyphonenumber = 2732133

**Query # 3**

http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=\*:\*&fq={!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company

**Response # 3**

<?xml version="1.0" encoding="UTF-8"?>

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">1</int>

<lst name="params">

<str name="q">\*:\*</str>

<arr name="fq">

<str>{!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company</str>

<str>{!parent which='type\_s:record'}consumerstate:MA AND type\_s:consumer</str>

</arr>

</lst>

</lst>

<result name="response" numFound="2" start="0">

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123</str>

<str name="type\_s">record</str>

<long name="\_version\_">1520747813059690496</long>

</doc>

<doc>

<int name="recordid">567</int>

<str name="referencenumber">567</str>

<int name="intreferencenumber">567</int>

<str name="documentid">567</str>

<str name="type\_s">record</str>

<long name="\_version\_">1520838033847353344</long>

</doc>

</result>

</response>

## **Use Case # 4**

Trying to find records with primarycompanyareacode = 781 and primarycompanyphonenumber = 2732133 and consumercity = Boston and return all fields of parent and child documents

**Query # 4**

http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=\*:\*&fq={!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company&fq={!parent which='type\_s:record'}consumercity:Boston AND type\_s:consumer&fl=\*,[child parentFilter=type\_s:record ]&wt=xml&indent=true

**Response # 4**

<?xml version="1.0" encoding="UTF-8"?>

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">2</int>

<lst name="params">

<str name="fl">\*,[child parentFilter=type\_s:record ]</str>

<str name="indent">true</str>

<str name="q">\*:\*</str>

<str name="wt">xml</str>

<arr name="fq">

<str>{!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company</str>

<str>{!parent which='type\_s:record'}consumercity:Boston AND type\_s:consumer</str>

</arr>

</lst>

</lst>

<result name="response" numFound="1" start="0">

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123</str>

<str name="type\_s">record</str>

<long name="\_version\_">1520747813059690496</long>

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123C1</str>

<arr name="primarycompanyname">

<str>ABB</str>

</arr>

<arr name="primarycompanyareacode">

<str>781</str>

</arr>

<arr name="primarycompanyphonenumber">

<str>2732133</str>

</arr>

<str name="type\_s">company</str></doc>

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123C2</str>

<arr name="primarycompanyname">

<str>ATT</str>

</arr>

<arr name="primarycompanyareacode">

<str>781</str>

</arr>

<arr name="primarycompanyphonenumber">

<str>2732233</str>

</arr>

<str name="type\_s">company</str></doc>

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123CM1</str>

<arr name="consumercity">

<str>Boston</str>

</arr>

<arr name="consumerstate">

<str>MA</str>

</arr>

<arr name="consumercountry">

<str>USA</str>

</arr>

<str name="type\_s">consumer</str></doc></doc>

</result>

</response>

## **Use Case # 5**

Trying to find records with primarycompanyareacode = 781 and primarycompanyphonenumber = 2732133 and consumercity = Boston and return all fields of parent document and all fields of company document but not any other child docuemnts

**Query # 5**

http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=\*:\*&fq={!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company&fq={!parent which='type\_s:record'}consumercity:Boston AND type\_s:consumer&fl=\*,[child parentFilter=type\_s:record childFilter=type\_s:company]&wt=xml&indent=true

**Response # 5**

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">2</int>

<lst name="params">

<str name="fl">\*,[child parentFilter=type\_s:record childFilter=type\_s:company]</str>

<str name="indent">true</str>

<str name="q">\*:\*</str>

<str name="\_">1450387801939</str>

<str name="wt">xml</str>

<arr name="fq">

<str>{!parent which='type\_s:record'} primarycompanyareacode:781 AND primarycompanyphonenumber:2732133 AND type\_s:company</str>

<str>{!parent which='type\_s:record'}consumercity:Boston AND type\_s:consumer</str>

</arr>

</lst>

</lst>

<result name="response" numFound="1" start="0">

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123</str>

<str name="type\_s">record</str>

<long name="\_version\_">1520747813059690496</long>

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123C1</str>

<arr name="primarycompanyname">

<str>ABB</str>

</arr>

<arr name="primarycompanyareacode">

<str>781</str>

</arr>

<arr name="primarycompanyphonenumber">

<str>2732133</str>

</arr>

<str name="type\_s">company</str></doc>

<doc>

<int name="recordid">123</int>

<str name="referencenumber">123</str>

<int name="intreferencenumber">123</int>

<str name="documentid">123C2</str>

<arr name="primarycompanyname">

<str>ATT</str>

</arr>

<arr name="primarycompanyareacode">

<str>781</str>

</arr>

<arr name="primarycompanyphonenumber">

<str>2732233</str>

</arr>

<str name="type\_s">company</str></doc></doc>

</result>

</response>

# **Initial Indexing**

Indexing logistics in SOLR can inherit a lot in terms of database configurations and processes from existing FAST indexing. We may continue to use the identical load tables from which Flume agent can crawl the records. In the SQL Source of the Flume we make use of a select statement similar to the one that follows-

Select documentid, documented as RecordId, referencenumber, convert(int, referencenumber) as intreferncenumber, pcxml as primarycompany, conxml as consumerxml, convert(varchar(33), createddate, 126) + ‘Z’ as createddate

Important to note that we do not need to roll up and make use of child entity fields such as Company, Consumer, fields (as multivalue fields). Scope XML fields can be used to capture and index information from child entity. As a result of that, we could do away with substantial DB processing during the loading of records.

Flume source is configured with a batch size properties. The source pulls in records up to the batch size in one go and push it to the channel. Sinks process these records and adds them to the index through Morphlines. Typically rate of processing at the Sink is slower than at the Source. Channel acts as buffer in this case.

Flume Sink sends back the status of completion to the channel and channel in turn communicates to the source for next batch pull. That way no additional support is needed from the database side. In fact no differential treatment is needed for incremental indexing. Same table can continue to serve as incremental load table.

# **Incremental Indexing**

Flume service once instantiated runs continuously and streams data as new data becomes available. Flume makes use of Incremental Column for keeping track of records already streamed and next record to be streamed. To take advantage of this feature, the incremental table is added with an Identity column (Id). The flume configuration that indicates the incremental column is as follows:

*# Increment column properties*

*agent1.sources.sql-source.incremental.column.name = id*

*# Increment value is from you want to start taking data from tables (0 will import entire table)*

*agent1.sources.sql-source.incremental.value = 0*

As Flume is done streaming all records from the incremental load table, it keeps a record of what the last Id that was crawled. It waits for further records with following criteria

*16/01/18 19:52:48 INFO source.SQLSource: Query: SELECT \* FROM FAST-CISMain-Updates WHERE id>100 ORDER BY id;*

As soon as another record gets added to the table, the streaming event gets fired and Flume executes the query to pull the record. This mechanism helps us achieving what we call Near Real Time indexing.

# **Home Page Search Queries**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Search URL** | **Query Syntax** |
| Search for Created Date within lower and upper limits | [http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=createddate:[2011-01-01T00:00:00.000Z TO 2014-01-01T00:00:000Z]](http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=createddate:%5b2011-01-01T00:00:00.000Z%20TO%202014-01-01T00:00:000Z%5d)&fl=\*,[child parentFilter=type\_s:record childFilter=type\_s:company] | createddate:[2011-01-01T00:00:00.000Z TO 2014-01-01T00:00:000Z] |
| Search for Created Date with min and upper limit | [http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=createddate:[\* TO 2014-01-01T00:00:000Z]](http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=createddate:%5b*%20TO%202014-01-01T00:00:000Z%5d)&fl=\*,[child parentFilter=type\_s:record childFilter=type\_s:company] | createddate:[\* TO 2012-01-01T00:00:000Z] |
| Search for Company Name | [http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanyname:"bank of america"](http://ftcdevsolr01.ppl.local:11000/solr/Test/select?q=primarycompanyname:%22bank%20of%20america%22)&fl=\*,[parent parentFilter=type\_s:record childFilter=type\_s:company] | primarycompanyname:"bank of america" |
| Search for Company City | [http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanycity:"OMAHA"](http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanycity:%22OMAHA%22)&fl=\*,[parent parentFilter=type\_s:record childFilter=type\_s:company] | primarycompanycity:"new york" |
| Search for Company State | [http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanystate:"California"](http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanystate:%22California%22)&fl=\*,[parent parentFilter=type\_s:record childFilter=type\_s:company] | primarycompanystate:"north carolina" |
| Search for Postal Code | [hhttp://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanyzip:"94304"&fl=\*,[parent parentFilter=type\_s:record childFilter=type\_s:company]](http://ftcdevsolr01.ppl.local:11000/solr/Test/select?q=primarycompanystate:%22north%20carolina%22) |  |
| Search for Consumer City | [http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumercity:"PALO ALTO"](http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumercity:%22PALO%20ALTO%22)&fl=\*,[parent parentFilter=type\_s:record childFilter=type\_s:consumer] | consumercity:"atlanta" |
| Search for Consumer State | [http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumerstate:"California"](http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumerstate:%22California%22)&fl=\*,[parent parentFilter=type\_s:record childFilter=type\_s:consumer] | consumerstate:"maryland" |
| Combination of fields | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=createddate:[\* TO 2014-08-26T00:00:00.000Z]&fq={!parent which='type\_s:record'}consumerstate:"California" AND type\_s:consumer&fl=\*,[child parentFilter=type\_s:record childFilter=type\_s:consumer] | createddate:[\* TO 2014-08-26T00:00:00.000Z] and consumerstate:"California” |

# **Proximity Search, Search Operators, Wild Card Character Search**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Search URL** | **Query Syntax** | **Comment** |
| Search for phrase | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumercity:"SAN DIEGO"&wt=xml&indent=true | consumercity:"SAN DIEGO" | Looking up for whole phrase |
| Search for wildcard at end (without space) | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumercity:SAN\*&wt=xml&indent=true | consumercity:SAN\* | This one shall match any word with leading "SAN" |
| Search for wildcard at end (with space) | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumercity:SAN \*&wt=xml&indent=true | consumercity:"bank \*" | This one shall match any phrase with leading word "SAN" |
| Search for wildcard at the beginning | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumercity:\*DIEGO&wt=xml&indent=true | consumercity:\*DIEGO | Should match word or phrase with trailing "DIEGO" . |
| Search for wildcard in the middle | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=consumercity:SAN\*GO&wt=xml&indent=true | Consumercity:SAN\*GO | Returns matches like SAN DIEGO |
| Proximity Search (unordered) | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanyname:"Industrial CHina" ~4&wt=xml&indent=true | primarycompanyname:"Industrial CHina" ~4 | Search for record where "Industrial" and "CHina" are within 4 words of each other. Returns **Industrial and Commercial Bank of CHina**  Less than 4 does not return a match |
| Proximity Search (ordered) |  |  | Ordered proximity search (as ONEAR in FAST) is not supported |
| Boolean Search - AND | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanyname:("Industrial" AND "Commercial")&wt=xml&indent=true | primarycompanyname:("Industrial" AND "Commercial") | Returns **Industrial and Commercial Bank of CHina** |
| Boolean Search - OR | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanyname:("Industrial" OR "Commercial")&wt=xml&indent=true | primarycompanyname:("Industrial" OR "Commercial") | Returns **Industrial and Commercial Bank of China** and **Commercial Recovery** |
| Fuzzy Search | http://ftcdevspark01.ppl.local:8983/solr/ScopePOC/select?q=primarycompanyname:"Industreal" ~2&wt=xml&indent=true | primarycompanyname:"Industreal" ~2 | Does not work. Expected to return all name with “Industrial” but does not. Not a reliable option. |

# **Time Dimension Navigator**

Time Dimension Navigators are representation of Date Ranges in more readable format. Examples of such date ranges would be –

* + This Week
  + This Month
  + This Year

Internally they are represented by Date Ranges such as

* + This Week => [NOW-7DAY/DAY TO NOW/DAY]
  + This Month => [NOW-1MONTH/MONTH TO NOW/MONTH]
  + This Year => [NOW-12MONTH/MONTH TO NOW/MONTH]

These filters are applied as part of facet query along with query as below

facet.query=createddate:[NOW-7DAY/DAY TO NOW/DAY]

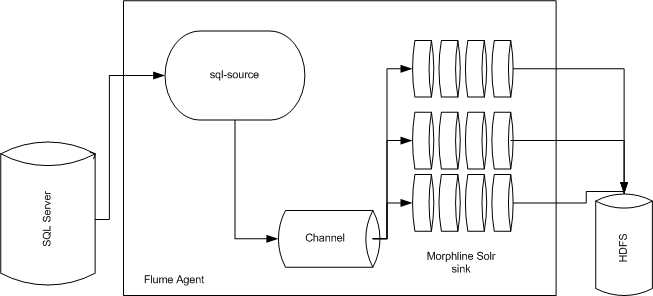
facet.query=createddate:[NOW-1MONTH/MONTH TO NOW/MONTH]

Multiple facet queries can be applied to a single query.

The response that returns, includes the all facet range values and corresponding result count if any. Application needs to identify the facet and apply the label such as “This Week”, “This Month” etc.

# **Performance Improvement**

Now flume is taking around 5 mins to create SOLR index with 10000 records. The main bottleneck in SOLR Morphline is the sink. Here sink is processing one record at a time, there is no way to configure a batch processing. But we can improve the performance by creating multiple sink. Flume support multiple sink reading from same channel and working on them. In this case our approach would be one SOLR-source writing to memory channel and then multiple SOLR-Sinks(say 3) are reading from the channel and creating SOLR index.



# **Appendix**

## Schema.xml



## Solrconfig.xml

****