A Default Architecture for incorporating Vector Space Search to the Enterprise

- An Open Source Perspective

Praseed Pai K.T.
Services and Solutions
UST Global Inc.
Trivandrum, Kerala
India

Shine Xavier
Emerging Accounts
UST Global Inc.
Trivandrum, Kerala
India

***Abstract*— Amazon Elastic Search and Apache Solr are the world's leading Open source Search systems. Both are powered by Lucene Engine. These systems can search large volume of texts, rank and sort the items based on its relevance. All modern search system uses Vector space model and these systems search within the index information created by a front end program. All of these tools give excellent command line tools which helps us to extract the full text Index from the documents. Too often, a full text search on documents will result in information "Cacophony" and search results might not reflect the actual semantics of the document. A document index generation system which can be programmed to be context aware can result in better search results. The index generated by such a program can be used for Feature Extraction. This paper chronicles such a system we prototyped by leveraging Apache Lucene and other content manipulation libraries like Apache Tika, Apache POI and iText.**

Keywords— Elastic Search, Document index, Feature, content manipulation, Apache, Solr,Open Source

I. INTRODUCTION

A large number of companies are leveraging Elastic Search or Systems build on Apache Solr for their domestic search requirements. For any search system to function, Indexing the document properly is a pre-requisite and the tools accompanying these search stacks do full text indexing in a context free manner. These tools extract the document physical attributes and the entire content will be included in the index as a field. This can result in noise "creeping" into the system and relevance of the search results might not be in the order by which the system has retrieved the results. We are proposing a context specific document text processing layer and reference architecture of solution for the generation of Indexes before the documents are persisted in the content repository.

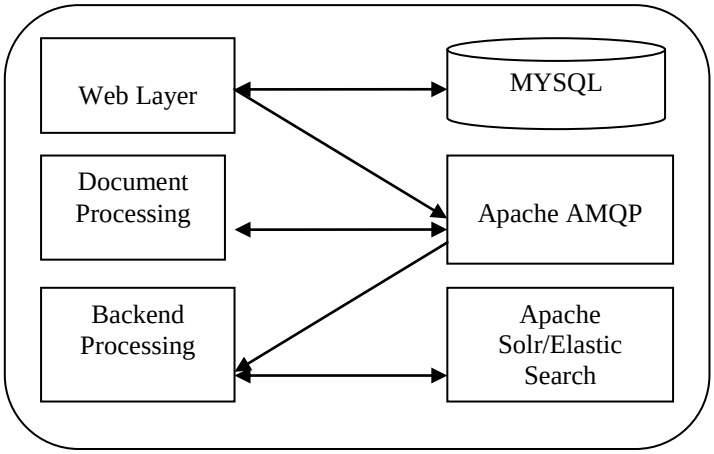
II. THE PROBLEM

Full text indexing will result in poor retrieval of the information and the index database cannot be used as repository for ad-hoc analytics. A content based feature extraction is a challenge as document templates do change on an ad-hoc basis. Trying to use a document search index for the purpose of analytics mandates a system which needs to be tweaked repeatedly until necessary feature vectors are stabilized. This can be done only on an empirical basis, after the system has moved to production

III. SOLUTION

A Solution to the above problem was built by following a Layered Architecture consting of Web Layer, Document Processing Layer and the Backend. The integration between the system is done using a Messaging system (AMQP) and Shared Folders. We packaged each sub system into Executable Jars. These sub systems can be indepdently triggered and the only pre-requisite is a JMS compliant system with pre-defined Queues and Topics. A high level architecture diagram is given below:

Fig. I

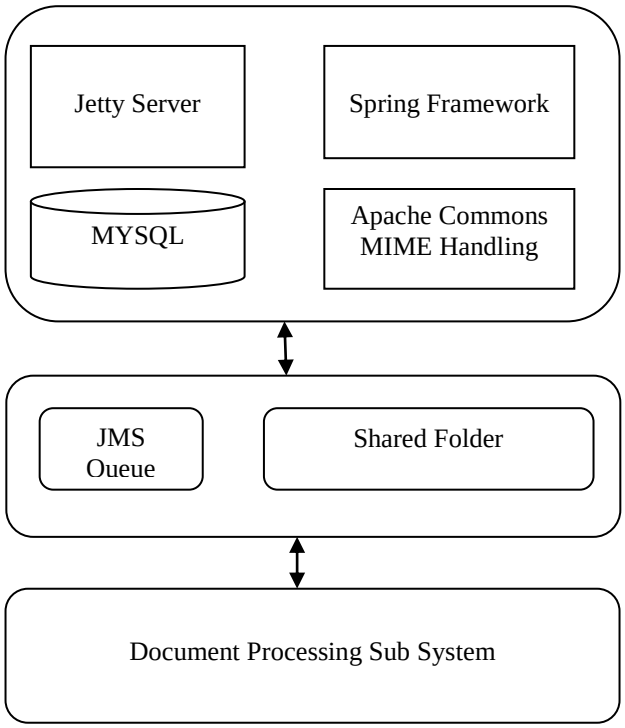


IV. Sub Systems

A. *Web Frontend Sub Sytem*

To host the Document harnessing engine, Jetty Server, an Embeddable Servlet engine was the platform of the choice. To handle HTML file uploads, the framework used Spring framework and Apache commons on the server side. The file name was mangled with a universally unique identifier (UUID) to avoid name collision while saving the file to a shared folder. The Meta data of the file is serialized into a JDBC compliant Database with a pre-defined Schema. For the reference implementation, we have used MYSQL, an Industrial strength database engine. Once the file has been saved into the shared folder and Meta data persisted in the database, a JMS message containing the Meta data is posted into the messaging queue. All the communications are using a Publish-Subscribe idiom of communication.

Fig. II



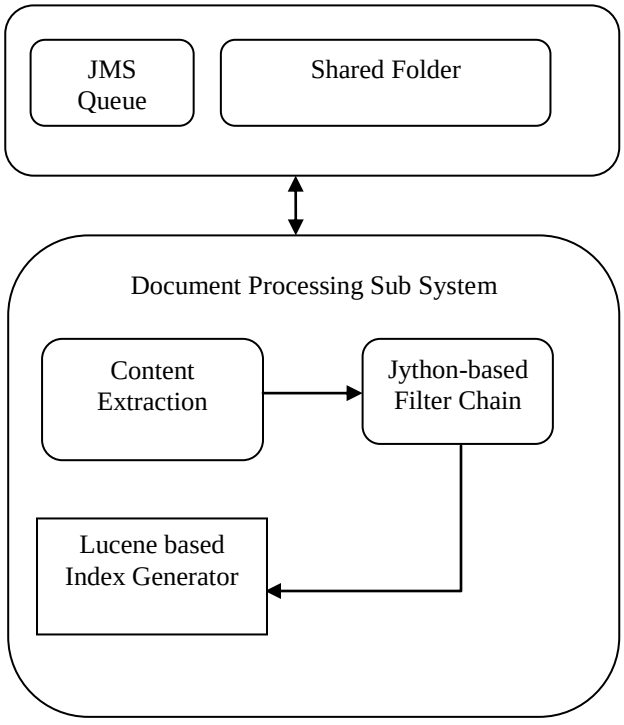
B. *Document Processing Sub System*

The framework can support myriad file formats which are in vogue across the world. For a system of this nature, it should support Microsoft Word, Microsoft Excel, Adobe PDF, Microsoft PowerPoint and imaging formats like JPG, PNG, TIFF (Content extracted through OCR system). The following table gives the prevalent formats and how to extract contents.

File Extensions	Library	Remarks
.DOC/.DOCX .PPT .XLS	Apache POI	Apache POI project handles most of the Document formats from Microsoft
.PDF	iText	Industrial strength PDF handler
.JPG .PNG .TIFF	Apache Tikka Tesseract	Apache Tikka is useful for identifying document types, Tesseract is an Open Source OCR Engine

To create Indexes which are relevant for the document context, we incorporated Jython Interpreter for enabling Python scripts to customize the text filtering process. We also created a minimal document processing workflow engine and an expression evaluator to trigger the transition between various stages in the workflow. After generating the Keywords in the context, we used Apache Lucene API to generate Index documents to be populated in the Backend System. After the generation of the Index, the Index document will be put in a shared folder and a JMS message will be posted

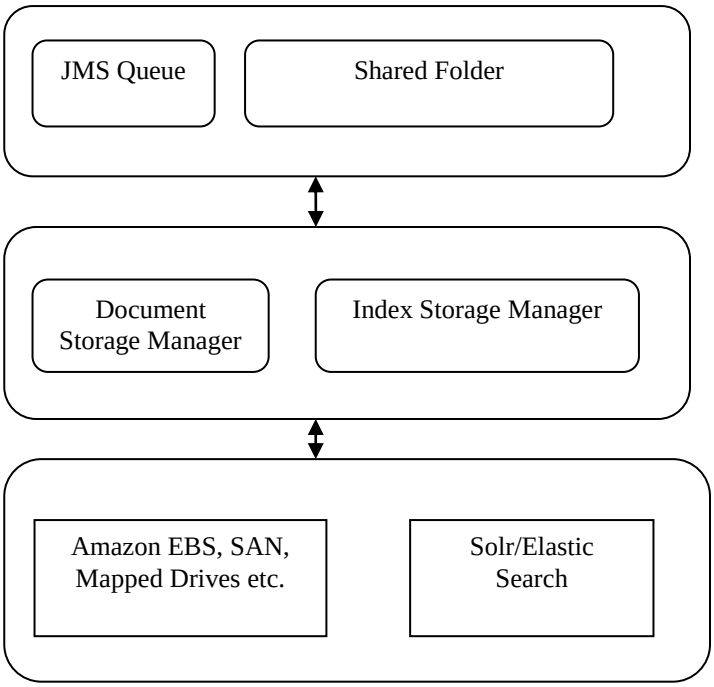
Fig. III



C. The Backend Processing Sub System

The Backend processing Sub system is integrated with the Document processing system through JMS Publish-Subscribe idiom. The notification will contain the information about the source document and the Index file. The backend system contains a Storage Manager Plugin to take care of the Backend Store which can be Storage Area Network (SAN), AWS Storage etc. The Index storage manager is leveraged to upload the document to Apache Solr (using SolorJ), Elastic Search, Amazon Elastic Search etc. Once the document is stored and Index posted, a callback REST service will made to the Web Layer to indicate the finishing of the whole process. The Web Layer will update the database and do the necessary clean up activity in the shared folder.

Fig. IV



V. INTEGRATION

Apache AMQP is the server system we have used for the Messaging sub system. To avoid polling, we used Publish-Subscribe Idiom. From an integration standpoint, the framework uses Shared files,Point-to-Point integration through Web Services and Asynchronous messaging.

VI. CONCLUSION

Apache Lucene is a project which uses Vector Space approach to textual Indexing and search. By leveraging Lucene as an engine, Search systems like Solr, Elastic Search, AWS Elastic search are helpful in realizing the dream of having an Open Source Search Engine. Index generation is a process which cannot be solved generally and every enterprise require some kind of context aware front end to populate the Index information corresponding to each document fed into the system. The Architecture outlined here helps to achieve the objective of having a robust front end for search systems.

ACKNOWLEDGEMENTS

Authors wish to thank various people for their contribution to this paper; Mr. Varghese Cherian, Director, Services and Solutions, UST global for helping us to publish this to a forum.

REFERENCES

[1] Gregory Hohpe and Bobby Woolf, ”Enterprise Integration Patterns” Addison-Wesley, pp. 39-97, 2004

[2] Eric Gama et al, “Design Patterns – Elements of Reusable Object Oriented Software”, pp. 1-70, 1996