Enterprise Search Engine

Minor project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology
in
Computer Science and Engineering
by

Amitosh Swain Mahapatra	(1501106221)
Asutosh Panda	(1501106503)
Manaswini Das	(1501106511)
Pratik Sanganeria	(1501106515)

Under the guidance of

Mrs. Arpita Majhi
Lecturer, Dept. of CSE,
CET, Bhubaneswar

Department of Computer Science and Engineering
College of Engineering and Technology
Bhubaneswar

Certificate

This is to certify that the project entitled **Enterprise Search Engine** is submitted by *Amitosh Swain Mahapatra, Asutosh Panda, Manaswini Das and Pratik Sanganeria* bearing registration numbers *1501106221, 1501106503, 1501106511, 1501106515* respectively to the *Department of Computer Science and Engineering, College of Engineering and Technology.* It is a record of bonafide research work to award partial Bachelor's Degree in Technology in Computer Science and Engineering under *Biju Patnaik University of Technology, Rourkela, Odisha.*

Date:

Dr.Subasish Mohapatra
HOD, CSE
CET, Bhubaneswar

Mrs. Arpita Majhi
Lecturer, Dept. of CSE
CET, Bhubaneswar

Declaration

We, declare that the work contained in thesis is original and has been done by us under the general supervision of our supervisor. The work has not been submitted to any other institute for any degree or diploma. We have followed the guidelines provided by the institute in writing the report. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the institute. Whenever we have used materials (data, theoretical analysis, figures etc) from the other sources, we have given due credits to them by citing them in the text of the report and giving their details in the references. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required detail references.

Amitosh Swain Mahapatra
Regd. No: 1501106221

Asutosh Panda
Regd. No: 1501106503

Manaswini Das
Regd. No: 1501106511

Pratik Sanganeria
Regd. No: 1501106515

Acknowledgement

It is our privilege and solemn duty to express our deepest sense of gratitude to Mrs. Arpita Majhi, Lecturer at the Department of Computer Science, under whose able guidance we carried out this work. We are indebted to her for his invaluable supervision, heart full cooperation, timely aid and advice till the completion of the report in spite of her pressing engagements.

We wish to record our sincere gratitude to Dr. Subhasish Mohapatra, Head of Department, Department of Computer Science and Engineering, for his constant support and encouragement in preparation of this report.

We take this opportunity to express our hearty thanks to all those who helped ours in the completion of our project work. We are very grateful to the author of various research papers, for helping us become aware of the research currently ongoing in the field.

We are very thankful to our parents for their constant support and love. Last, but not least, we would like to thank our classmates for their valuable comments, suggestions and unconditional support.

Abstract

Like libraries, corporations, government agencies, and not-for-profit organizations have to deal with documents in many different media and formats, but much of that information is unique and proprietary to the organization. Some of an organization's information assets may be held in storage systems, relational databases or specialized applications, but much is unstructured text of the type information retrieval systems have been designed to deal with. A prudent enterprise search system would allow the easy retrieval of all such documents. We present a design of a system for indexing and searching through the large cache of documents scattered around heterogeneous systems accessible in the internal network of the organization, with a robust access control system.

Contents

List of Figures

Introduction

Organizations generate a huge amount of data. Reports, communications, budgets, specifications, memos and invoices are all text content. This application of information retrieval technology to information finding within organisations has become known as "enterprise search". Enterprise search may be interpreted as search of digital textual materials owned by an organisation, including search of their external Website, company intranet, and any other electronic text that they hold such as email, database records and shared documents. Many characteristics of enterprise search represent a significant challenge for IR system designers.

Information in the enterprise may be structured or unstructured. Documents are produced by a variety of sources, perhaps in many different languages, and generally without formatting standards. Metadata may be created according to a number of different schemes, or may not be added at all. Not all users have the same access rights to all information, and some information, such as employee records, are highly confidential. The need to federate across different repositories of information means that a single ranked list must be created for data from a variety of sources and formats.

Different contexts may require different ranking methods. That is, search tools for the enterprise must perform many functions in addition to simple indexing and query processing. A basic search tool will not do. There is an expectation, based on experience with the Web, that finding corporate information should be fast and efficient, and that it should be done through a single interface. However within organisations these expectations have typically not been met, and there is evidence that employees spend a significant amount of their time searching for, and often failing to find, information needed to perform their work.

For instance,

1. According to IDC, a company with 1,000 information workers can expect more than $5M in annual wasted salary costs because of poor search. They report that people spend 9-10 hours per week searching for information and aren't successful from one-third to half of the time.

2. According to Butler Group, as much as 10% of a company's salary costs are wasted through ineffective search.

3. A 2007 Accenture survey of 1,000 middle managers found they spend as long as two hours a day searching for information and that more than half the information they find during searching is useless.

Another area of high financial impact for enterprise search is the area of e-discovery. Search tools capable of searching all the sources of information within an organizations are in increasing demand for supporting discovery actions associated with high-value lawsuits, even if these searches are conducted by external professionals.

Further, effective search on an organizations's outward-facing Websites can be critical to an organization's mission — be it disseminating information, supporting government campaigns, matching job applicants to job vacancies, or generating online sales. E-commerce sites are often driven by a search tool whose functions include supporting search for product information and reviews, location of the actual product purchase page, search-driven advertising and intelligent recommendations.

Consequently, enterprise search software on external websites is a highly valued source of information about the interests of customers and stakeholders. Further, query data they generate can give information about trends and sudden spikes in customer/community interest as well as identifying unmet demand. Indeed, leading enterprise search tools provide extensive reporting capabilities.

Enterprise search tools perform other functions too. Navigation links, RSS feeds, faceted browsing and taxonomy displays on organizations Websites are often powered by search engines. Search tools are increasingly used within organizations to locate expertise when putting together project teams.

A Review of Existing Solutions

Enterprise search software has been available for some time, and some of the earlier systems were spin-offs from academic research in information retrieval. Companies such as FAST Search & Transfer (acquired by Microsoft in 2008) and Autonomy (which acquired Verity in 2005 and the Interwoven content management system (CMS) technology in 2009) are well-known for their enterprise search.

2.1 Google Search Appliance

Google Search Appliance (GSA), a now discontinued Google product, was the most popular enterprise search solution in this sector. This was partly due to the Google brand and the ease of setup and use of the solution. GSA was a complete hardware plus software stack that provided document indexing facilities. It ran on an operating system based on CentOS and Dell PowerEdge hardware. However, the major inconvenience with the system was it being a black box to end-users and inflexibility of hardware.

2.2 Custom Solutions

Other software companies like IBM and Oracle, Smaller companies offering enterprise search products may create a niche through special features. Vivisimo, developers of a search engine which provides clustered output, also have an enterprise search product for the corporate market. Endeca offers a product with "guided navigation" in which possible search filters are offered as part of the results screen. Funnelback specialises in "software as a service" (SaaS) for enterprise, Website and portal search. Various software solution providers also provide customized enterprise offerings which tailor to the specific needs of an enterprise. They develop the software on demand, with a detailed requirement specification, taken as traditional development projects.

2.3 Issues with existing systems

The main issue with existing solutions are the lack of integration options, restriction on customization and procurement cost. The process of designing a custom search engine is time consuming and difficult task especially when undertaken as a traditional software designing project. When medium and small scale organizations are presented with this decision, the costs outweigh the benefits. A simple, easy to use and readymade solution, built over open source components will be the perfect product for such organizations. Also, many organizations are not completely homogenous about their storage. Some rely on a cloud provider completely, and some choose a hybrid approach - keeping some data in cloud and some in on premise data stores. Some data are stored as blobs as part of existing systems such as SAP PI, ERP systems, CRM systems etc. The solutions available in the market make no effort in integrating with such systems and provide limited means of designing custom integrations. Custom needs are usually satisfied by customized offering from solution providers.

2.4 Cloud powered enterprise search

There has been a trend towards cloud powered enterprise search. Most of the enterprise utilize cloud storage in one form in other, so it makes a cost effective way to get started with cloud based search. However, agreements and data secrecy requirements are a hindrance to adoption of cloud based solutions. Also, a lot of the existing data may reside in local systems and cannot be immediately moved into cloud due to regulatory requirements. It is possible to access data from cloud from a corporate network but the vise-versa is not possible without significant changes to network. Thus, an on-premises solution has always been preferred.

Most of the available enterprise search products products, are based on Apache Lucene, an industry standard library for providing text processing and indexing capabilities. It provides the building blocks for building a search server, utilizing the lucene pipeline.

ElasticSearch and Apache Solr are two prominent search servers, built on the top of Lucene library. They provide the infrastructure to run Lucene in a networked and distributed fashion. It is important to know that a search server is not a search engine, even though it provides search and indexing capabilities.

A search server is similar to a document database with search capabilities, that lacks an UI and business logic. It is designed to be used by search engine

developers and not by end users.

A search engine is built around one or more search servers by implementing an UI, logic, ranking metrics and security. It is meant to be used directly by the end users. It takes care of the search server configuration to provide relevant and accurate results by configuring the preprocessing filters, query processing, ranking and result generation.

Project Requirements

Organizations vary by the amount of electronic text they produce. There are organizations that produce absolutely zero electronic text, however, in today's scenario, a typical medium scale organization produces about a quarter million pieces of content each year. Large organizations like IBM have reported to produce around fifty million pieces of content in one year. Content can be emails, letters, reports, IMs and automated text generated by machines. Keeping track of content at any scale is always a challenge. Adapting terminology from big data processing, We can define our goal is to conquer huge quantity of polymorphic text with variable degree of importance. We define "conquering" as successfully discovering, indexing and making the content search-able through the search engine, while respecting the organizational bounds at a minimal investment.

3.1 Scalability

From several hundred PDFs to a million automated reports, the search server should be scalable enough to ingest the influx of data. Vertical scalability in memory buffers and horizontal scaling with shredding.

3.2 Variety

Enterprises uses Emails, chats, PDFs, images, scans, invoices, excel sheets and data in databases. Such a wide variety of data is needed to be indexed for efficient search. We need to extract text during data acquisition. There are some objects need to be converted to text such as records from database tables and images so as to increase the search efficiency.

3.3 Heterogenous sources

Enterprise use heterogeneous sources like local servers, Google Drive, Amazon S3 and Relational Databases. We need to discover all data from various available sources in the Enterprise.

3.4 Ubiquity

Employees should be able to query information conveniently from any device, anywhere within the network having the proper access rights. This is provided by using a rich platform-independent search client such as a web app.

3.5 Security

Security features are robust and simple, using a common Username-Password standard and Single Sign-on. It also take some measures to prevent unauthorized access to employees as well as people outside the network.

3.6 Confidentiality

Confidentiality is ensured if the data is processed inside the network preferably. Data acquisition and crawling should be done in a controlled manner only within the system network by authorised personnel only.

3.7 Access control

Users should only see what they can assess from normal workflow. All the information are not useful to some set of users. Other than that, there are some sensitive data that should not be available to common employees. Ease of configuration and deployment The Enterprise Search Engine is very flexible and easily configurable according to organisation requirements. And also it is very easy to deploy in the network.

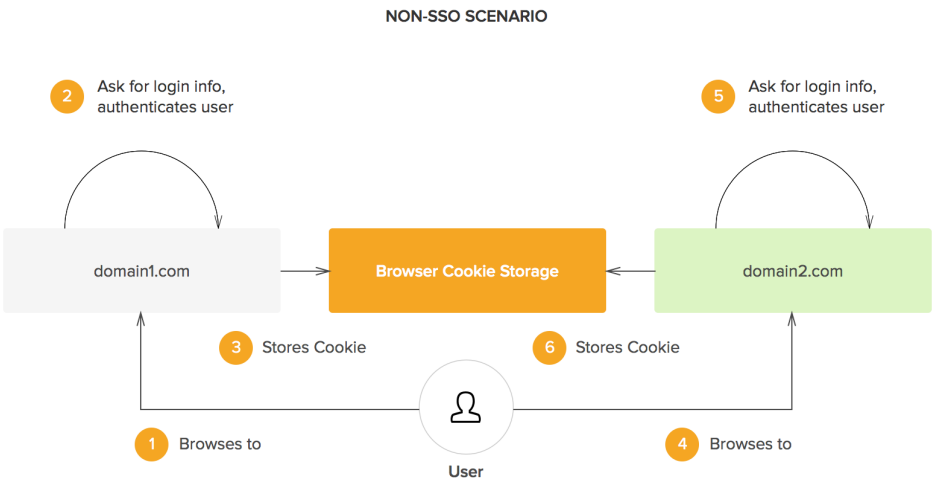

Figure 3.1: Single Sign-on

3.8 Single Sign On

This is provided in order to integrate with the existing systems without signing in again and again. Single sign-on (SSO) is a session and user authentication service that permits a user to use one set of login credentials (e.g., name and password) to access multiple applications. The service authenticates the end user for all the applications the user has been given rights to and eliminates further prompts when the user switches applications during the same session. On the back end, SSO is helpful for logging user activities as well as monitoring user accounts.

In a basic web SSO service, an agent module on the application server retrieves the specific authentication credentials for an individual user from a dedicated SSO policy server, while authenticating the user against a user repository such as a lightweight directory access protocol (LDAP) directory.

Some SSO services use protocols such as Kerberos and the security assertion markup language (SAML). SAML is an XML standard that facilitates the exchange of user authentication and authorization data across secure domains. SAML-based SSO services involve communications between the user, an identity provider that maintains a user directory, and a service provider. When a user attempts to access an application from the service provider, the service provider will send a request to the identity provider for authentica-

tion. The service provider will then verify the authentication and log the user in. The user will not have to log in again for the rest of his session. In a Kerberos-based setup, once the user credentials are provided, a ticket-granting ticket (TGT) is issued. The TGT fetches service tickets for other applications the user wishes to access, without asking the user to re-enter credentials.

3.9 Cost effectiveness

The Enterprise Search Engine employs the best use of open source which are freely available to explore. The servers are implemented on the top of by ElasticSearch which provides tools like analysers and filters to build an efficient and ideal Search Engine. The crawlers are built using simple programming language and it connects to the search servers via REST APIs. The security features are dependant on the Enterprise itself. No exclusive set of infrastructures are required like previous versions. So there is less or no cost involved in its creation and execution process.

All of the tools and libraries use standard and open source toolkits, ensuring no vendor lock-in

3.10 Extensible

One of the primary goals of enterprise search system is to allow search of diverse sources. Some of these are unique to the organization in question. The search system is expected to be extensible to allow independent addition of newer integrations. This should be guaranteed by providing a standard API for designing crawlers and client applications.

Implementation

4.1 ElasticSearch

Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License. Official clients are available in Java, .NET (C#), PHP, Python, Apache Groovy, Ruby and many other languages. According to the DB-Engines ranking, Elasticsearch is the most popular enterprise search engine followed by Apache Solr, also based on Lucene.

ElasticSearch forms the backend of the search engine, by providing a scalable and distributed search backend.

Elasticsearch is developed alongside a data-collection and log-parsing engine called Logstash, and an analytics and visualisation platform called Kibana. The three products are designed for use as an integrated solution, referred to as the "Elastic Stack" (formerly the "ELK stack").

Elasticsearch can be used to search all kinds of documents. It provides scalable search, has near real-time search, and supports multi-tenancy. "Elasticsearch is distributed, which means that indices can be divided into shards and each shard can have zero or more replicas. Each node hosts one or more shards, and acts as a coordinator to delegate operations to the correct shard(s). Rebalancing and routing are done automatically". Related data is often stored in the same index, which consists of one or more primary shards, and zero or more replica shards. Once an index has been created, the number of primary shards cannot be changed.

Elasticsearch uses Lucene and tries to make all its features available through the JSON and Java API. It supports facetting and percolating, which can be useful for notifying if new documents match for registered queries.

Another feature is called "gateway" and handles the long-term persistence of the index; for example, an index can be recovered from the gateway in the event of a server crash. Elasticsearch supports real-time GET requests, which

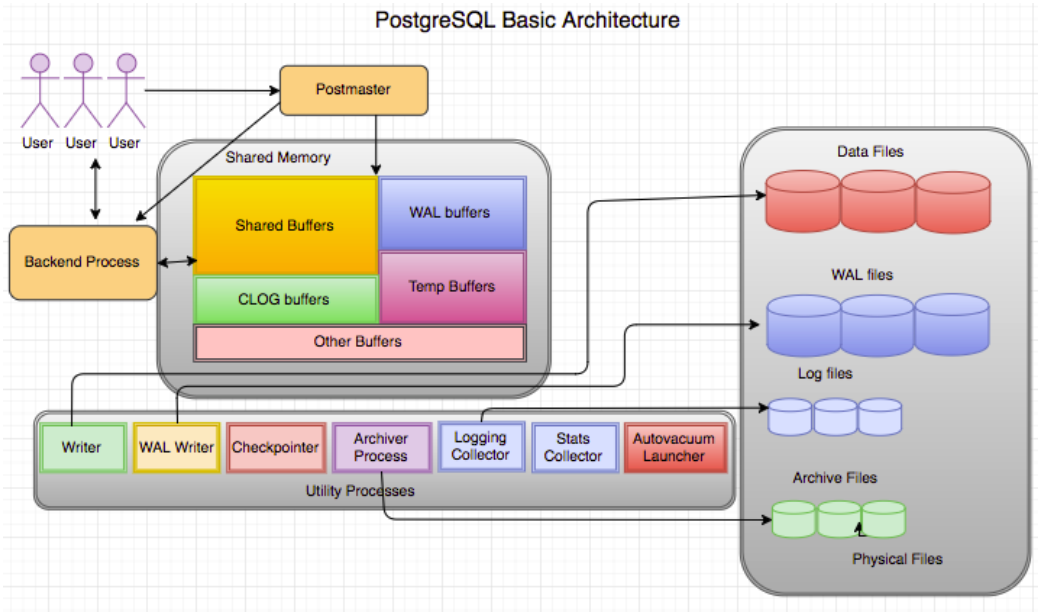

Figure 4.1: PostgreSQL Architecture

makes it suitable as a NoSQL datastore, but it lacks distributed transactions.

4.2 PostgreSQL

PostgreSQL is a general purpose and object-relational database management system, the most advanced open source database system. PostgreSQL was developed based on POSTGRES 4.2 at Berkeley Computer Science Department, University of California.

PostgreSQL was designed to run on UNIX-like platforms. However, PostgreSQL was then also designed to be portable so that it could run on various platforms such as Mac OS X, Solaris, and Windows.

PostgreSQL is free and open source software. Its source code is available under PostgreSQL license, a liberal open source license. You are free to use, modify and distribute PostgreSQL in any form.

PostgreSQL requires very minimum maintenance efforts because of its stability. Therefore, if you develop applications based on PostgreSQL, the total cost of ownership is low in comparison with other database management systems. We utilize Postgres to store authentication and metadata information. NodeJS.

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). It is a platform built on Chrome's JavaScript runtime

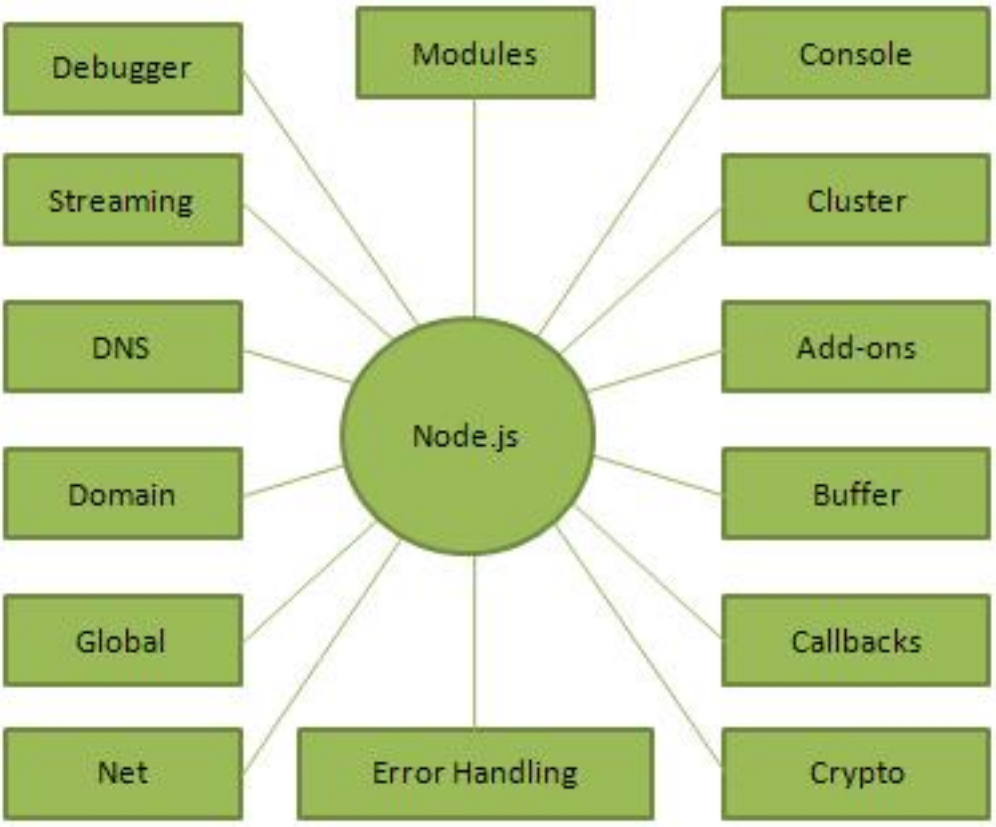

Figure 4.2: NodeJS

for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

4.3 JWT

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

We utilize JWT for authentication tokens. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

4.4 REST

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

A RESTful API – also referred to as a RESTful web service – is based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development.

A RESTful API breaks down a transaction to create a series of small modules. Each module addresses a particular underlying part of the transaction. This modularity provides developers with a lot of flexibility, but it can be challenging for developers to design from scratch. Currently, the models provided by Amazon Simple Storage Service, Cloud Data Management Interface and OpenStack Swift are the most popular.

A RESTful API explicitly takes advantage of HTTP methodologies defined by the RFC 2616 protocol. They use GET to retrieve a resource; PUT to change the state of or update a resource, which can be an object, file or block; POST to create that resource; and DELETE to remove it.

With REST, networked components are a resource you request access to – a black box whose implementation details are unclear. The presumption is that all calls are stateless; nothing can be retained by the RESTful service between executions.

4.5 SAML/WS-Federation

Security Assertion Markup Language is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. As its name implies, SAML is an XML-based markup language for security assertions (statements that service providers use to make access-control decisions).

The single most important use case that SAML addresses is web browser single sign-on (SSO). Single sign-on is relatively easy to accomplish within a security domain (using cookies, for example) but extending SSO across security domains is more difficult and resulted in the proliferation of non-interoperable proprietary technologies. The SAML Web Browser SSO profile was specified and standardized to promote interoperability. (For comparison, the more recent OpenID Connect protocol is an alternative approach to web browser SSO.)

WS-Federation (which is short for Web Services Federation) is a protocol that can be used to negotiate the issuance of a token. You can use this protocol for your applications (such as a Windows Identity Foundation-based app) and for identity providers (such as Active Directory Federation Services or Azure AppFabric Access Control Service).

You can also use the SamlConfiguration object (available in rules) to configure claims sent via the SAML token, as well as other lower-level WS-Fed and SAML-P settings.

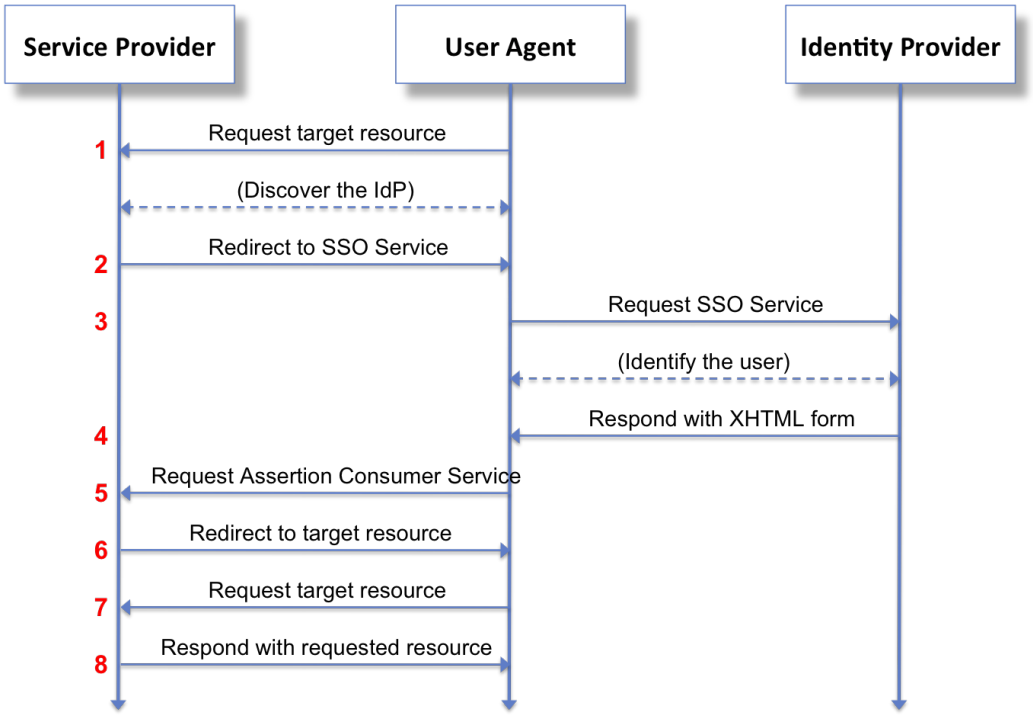

Figure 4.3: SAML

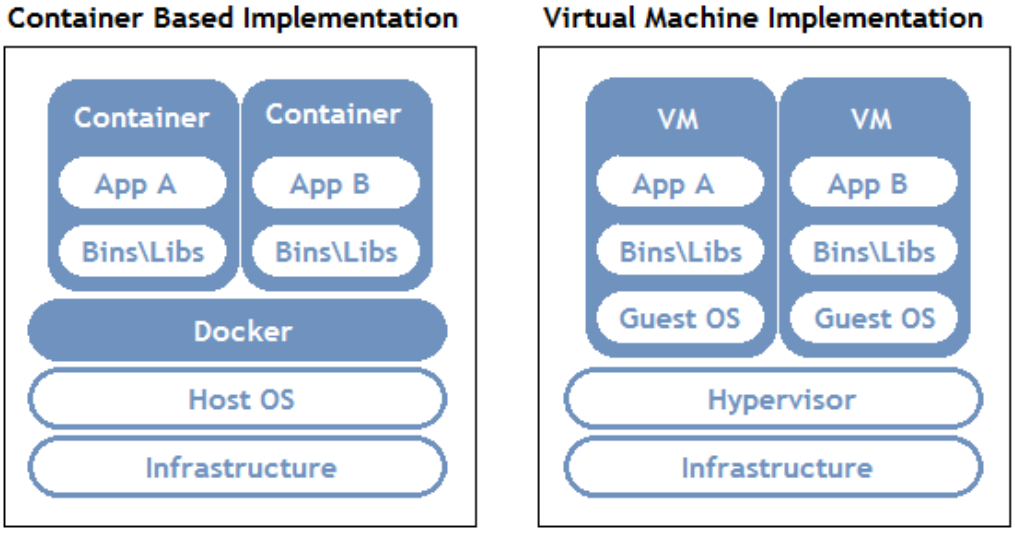

Figure 4.4: Docker Architecture

4.6 Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

4.7 ReactJS

ReactJS is an open-source JavaScript library which is used for building user interfaces specifically for single page applications. It's used for handling view layer for web and mobile apps. React also allows us to create reusable UI

components. React was first created by Jordan Walke, a software engineer working for Facebook. React first deployed on Facebook's newsfeed in 2011 and on Instagram.com in 2012.

React allows developers to create large web applications which can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in application. This corresponds to view in the MVC template. It can be used with a combination of other JavaScript libraries or frameworks, such as Angular JS in MVC.

4.8 API

We propose defining a standard API for search and indexing operations. It is a simple REST based API that allows the organization to program its own set of crawlers and integrate with its data store. We also provide a standard search API to aid in developing customized search clients.

Architecture

The above figure sketches the three broad phases involved in building a unified index of heterogeneous enterprise data: gathering, extracting, and indexing, as we now discuss.

5.1 Gathering Data

Crawler is a set of programs that visits (or crawls) through all type of data present in all the system of an Enterprise including data from local storages, relational and non-relational databases, network shares, cloud storages (Google Drive, Amazon S3, etc.).

Crawlers are typically programmed to visit each data sources that have been submitted by the owner or employees. The entire content of the data are visited and indexed. Crawlers gained the name because they crawl through the whole possible set of data items at a time, following the links to other documents in the whole system until all the documents have been read. The crawlers scans for any new document saved in the whole enterprise based database on a periodic basis and does the necessary stuffs to index the contents of the documents.

5.2 Data Extraction

5.2.1 Client side processing

The crawlers does the basic work of crawling through all sets of documents where the texts are extracted along with their absolute file location and stored in the form of a temporary dictionary. The dictionary is then sent to the search servers through REST where the indexing is done, employing a specific set of customisable analyzers and filters.

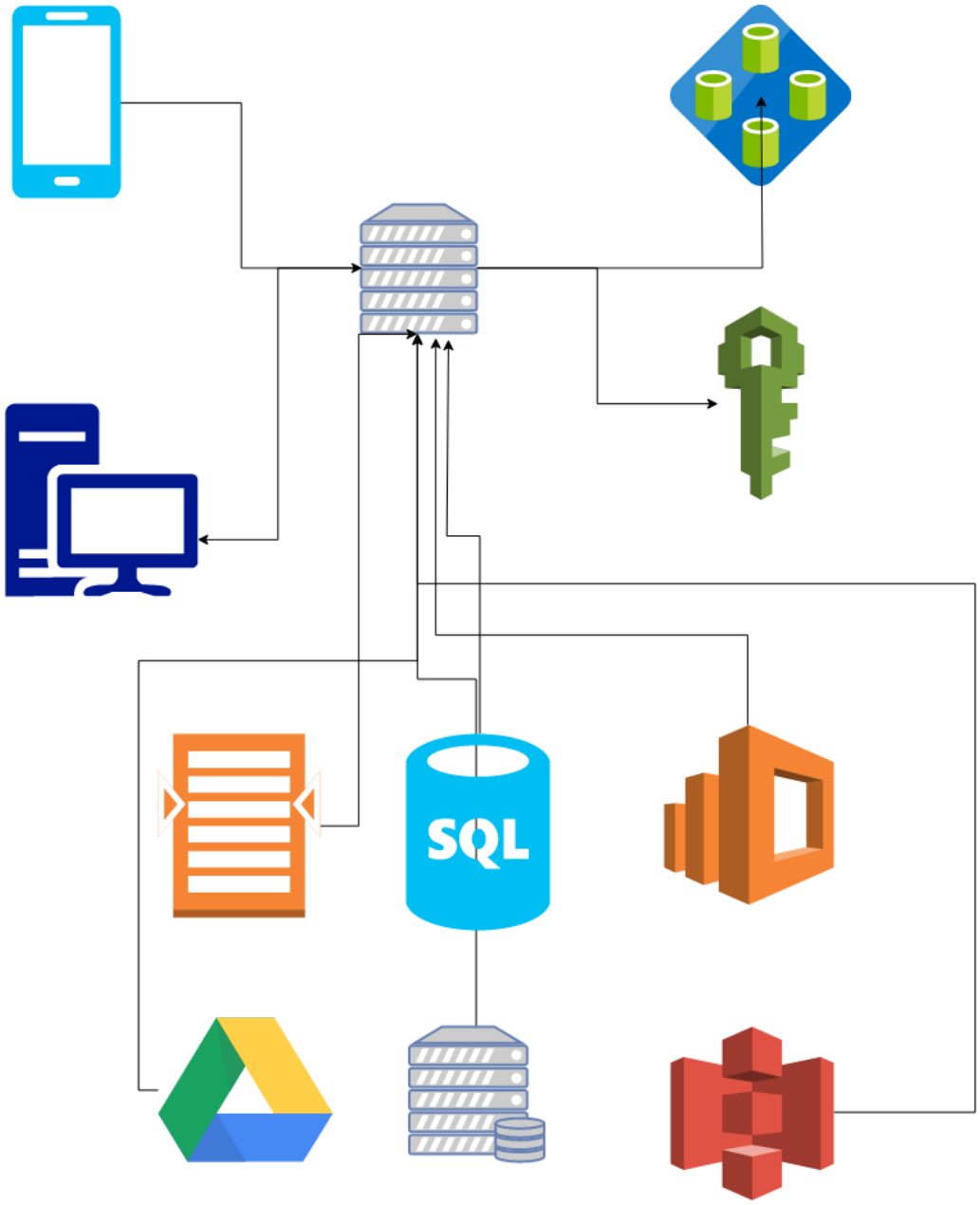

Figure 5.1: Architecture of proposed model

5.2.2 Server side processing

There are some of the built-in analyzers in the Elastic Search Servers, which are used to convert full-text strings into an inverted index, suitable for searching. The standard analyzer, which is the default analyzer used for full-text fields, is a good choice for most Western languages. It consists of the following:

1. The standard tokenizer, which splits the input text on word boundaries
2. The standard token filter, which is intended to tidy up the tokens emitted by the tokenizer (but currently does nothing)
3. The lowercase token filter, which converts all tokens into lowercase
4. The stop token filter, which removes stopwords—common words that have little impact on search relevance, such as a, the, and, is.

The standard analyser provides for a stopword filter. It is enabled by creating a custom analyzer based on the standard analyzer and setting the stopwords parameter. We can either provide a list of stopwords or tell it to use a predefined stopwords list from a particular language.

Stemmer

Stemming attempts to remove the differences between inflected forms of a word, in order to reduce each word to its root form. For instance foxes may be reduced to the root fox, to remove the difference between singular and plural in the same way that we removed the difference between lowercase and uppercase. The root form of a word may not even be a real word. The words jumping and jumpiness may both be stemmed to jump. It doesn't matter—as long as the same terms are produced at index time and at search time, search will just work. Elastic Search provides for default Stemmer Token filters for this purpose.

Analyzer

An analyzer examines the text of fields and generates a token stream. Analyzers are used both during ingestion, when a document is indexed, and at query time. An analyzer examines the text of fields and generates a token stream. Analyzers may be a single class or they may be composed of a series of tokenizer and filter classes. Although the analysis process is used for both indexing and querying, the same analysis process need not be used for both

operations. For indexing, you often want to simplify, or normalize, words. For example, setting all letters to lowercase, eliminating punctuation and accents, mapping words to their stems, and so on. Doing so can increase recall because, for example, "ram", "Ram" and "RAM" would all match a query for "ram".

To increase query-time precision, a filter could be employed to narrow the matches by, for example, ignoring all-cap acronyms if you're interested in male sheep, but not Random Access Memory. The tokens output by the analysis process define the values, or terms, of that field and are used either to build an index of those terms when a new document is added, or to identify which documents contain the terms you are querying for.

Filters

There are many filters provided by Elastic Search which are easily customizable according to the changing requirement. Some of the implemented filters are Numeric filters and date filters.

The numeric filter works by loading all the relevant field values into memory, and checking for the relevant docs if they satisfy the range requirements. This requires more memory since the numeric range data are loaded to memory, but can provide a significant increase in performance.If the relevant field values have already been loaded to memory, for example because it was used in facets or was sorted on, then this filter should be used.

In documents, dates are represented as strings. Elasticsearch uses a set of preconfigured formats to recognize and parse these strings into a long value representing milliseconds-since-the-epoch in UTC. It might be possible that date field might not be listed in the set of preconfigured ES date formats. Formatted dates will be parsed using the format specified on the date field by default, but it can be overridden by passing the format parameter to the range query.

Tokenizers

A tokenizer receives a stream of characters, breaks it up into individual tokens (usually individual words), and outputs a stream of tokens. For instance, a whitespace tokenizer breaks text into tokens whenever it sees any whitespace. It would convert the text "Quick brown fox!" into the terms [Quick, brown, fox!]. The tokenizer is also responsible for recording the order or position of each term (used for phrase and word proximity queries) and the start and end character offsets of the original word which the term represents (used for highlighting search snippets). Elasticsearch has a number of built in

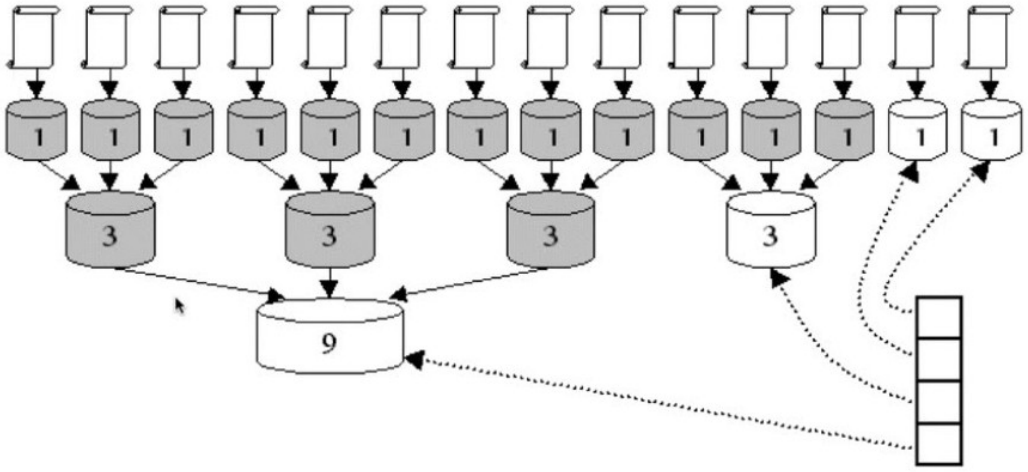

Figure 5.2: Lucene Architecture

tokenizers which can be used to build custom analyzers.

5.3 Indexing

Indexing can broadly be classified into three types when it comes to indexing in Enterprise Search.

We use periodic delta indexing in our project. Search engine indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. Index design incorporates interdisciplinary concepts from linguistics, cognitive psychology, mathematics, informatics, and computer science. Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Apache Lucene is an open source project. It has powerful features including scalable, high-performance indexing, cross-platform solutions and powerful, accurate and efficient search algorithms.

5.3.1 Indexing Documents

Document indexing consists of first constructing a document that contains the fields to be indexed or stored, then adding that document to the index. The key classes involved in indexing are, "oal.index.IndexWriter" which is

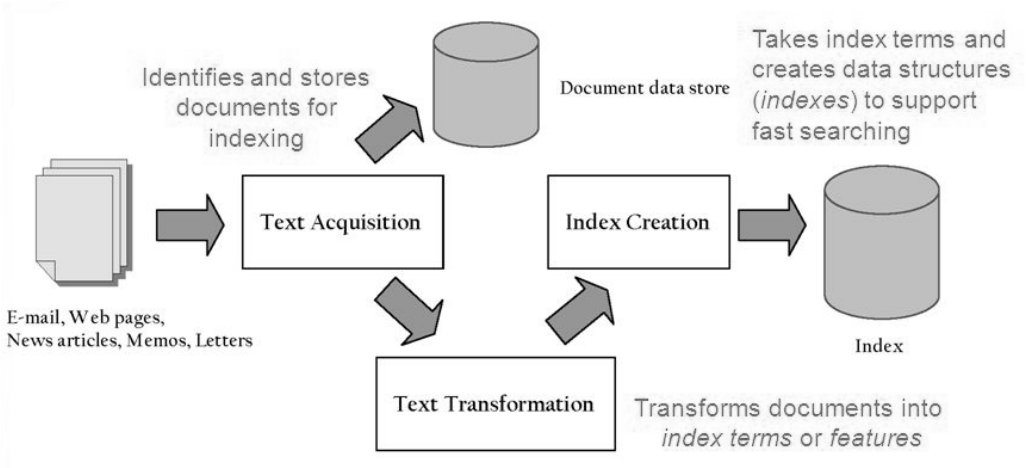

Figure 5.3: Lucene Indexing

responsible for adding documents to an index, and, "oal.store.Directory" which is the storage abstraction used for the index itself. Directories provide an interface that's similar to an operating system's file system. A Directory contains any number of sub-indexes called segments. Maintaining the index as a set of segments allows Lucene to rapidly update and delete documents from the index.

5.3.2 Document Search and Search Ranking

The Lucene search API takes a search query and returns a set of documents ranked by relevance with documents most similar to the query having the highest score. Lucene provides a highly configurable hybrid form of search that combines exact boolean searches with softer, more relevance-ranking-oriented vector-space search methods. All searches are field specific because Lucene indexes terms and a term is composed of a field name and a token.

Working

Our solution implements searching a distributed search index running on ElasticSearch. It provides faceted search, fuzzy matching, spelling correction, search suggestions and entity normalization. It also takes care of access control restrictions.

Web client, a ReactJS web app, is the primary reference implementation of the search user interface. It allows end-users to issue queries to search and displays the search result. The user is either presented with a login form or is automatically signed in when single sign-on in enabled.

Crawlers are independent programs that are used for content indexing. They are deployed to crawl one or more data sources periodically using operating system scheduling services.

Authentication is handled by the "Auth" layer. It uses a standard user-password model or a federated model when configured with SAML.

ACL design is based on extended UNIX permission model (SELinux). Each indexed object is assigned a type and group and each user belongs to a group. ACL Rules define which other groups can access the content.

In order to provide access to content over HTTP, each crawler defines an HTTP protocol proxy to allow download of content. It may either redirect to external service such as cloud provider or launch an associated application.

Finally, the entire application is packaged as a docker virtual appliance such that it can be deployed quickly without any additional configuration whatsoever.

6.1 How search works

A user wishing to perform a search visits the app using a device. If single sign-on is enabled, and an existing session ticket is available, the user is taken into the Search page, else they are presented with a Login screen. On successful login, the user is provided with a search input where they enter their search terms. This is passed as an API call to the backend using the

search API.

The backend receives the search API call and transforms the search terms to an ElasticSearch API call. Once ElasticSearch returns the results, it is filtered as per ACL rules. ACL is stored in a separate Postgres data store and identifiers are embedded in ElasticSearch records.

6.2 How crawling works

Crawlers are set to run periodically by the OS scheduler such as cron. It detects changes in metadata, permissions and content and pushes it to the server. Crawler use a key generated by the administrator to authenticate their requests to the search API.

Whenever an index request is received, the server validates the API key and the index object, calculates the ACL from file system metadata and indexes the content in ElasticSearch.

Future scope

7.1 Semantic Search

Web search engines are robust, efficient and very user-friendly. We intend to implement extraction of semantic data and answering questions that is commonly found in web search engines.

The word “semantic” refers to the meaning or essence of something. Applied to search, ”semantics” essentially relates to the study of words and their logic. Semantic search seeks to improve search accuracy by understanding a searcher's intent through contextual meaning. Through concept matching, synonyms, and natural language algorithms, semantic search provides more interactive search results through transforming structured and unstructured data into an intuitive and responsive database. Semantic search brings about an enhanced understanding of searcher intent, the ability to extract answers, and delivers more personalized results. Google's Knowledge Graph is a paradigm of proficiency in semantic search.

We intend to pursue the implementation of semantic search in our enterprise search solution. Using ML models designed as the part of DBPedia TextExt, we would extract semantic data from unstructured text and store using a graph database.

Later semantic queries will be evaluated against this graph database to generate results. Work in this regard has been done by Claire et al.

Conclusion

We discussed a design of a robust, cost-effective enterprise search solution. We used open source tools such as ElasticSearch, NodeJS and ReactJS to implement our idea to achieve the same. We explored the implementation of ElasticSearch and integration of crawlers in Amazon S3 and documents in local systems. We also accessed the performance, extensibility and stability of various existing systems and tried to remedy the major shortcomings in our design.

References