Lab 1. Introduction to Solr 7

Today we are in the age of digitization. People are generating data in different ways: they take pictures, upload images, write blogs, comment on someone's blog or picture, change their status on social networking sites, tweet on Twitter, update details on LinkedIn, do financial transactions, write emails, store data on the cloud, and so on. Data size has grown not only in the personal space but also in professional services, where people have to deal with a humongous amount of data. Think of the data managed by players such as Google, Facebook, the New York Stock Exchange, Amazon, and many others. For this data tsunami, we need the appropriate tools to fetch data, in an organized way, that can be used in various fields, such as scientific research, real-time traffic, fighting crime, fraud detection, digital personalization, and so on. All of this data needs to be captured, stored, searched, shared, transferred, analyzed, and visualized.

Analyzing structured, unstructured, or semi-structured ubiquitous data helps us discover hidden patterns, market trends, correlations, and personal preferences. With the help of the right tools to process and analyze data, organizations can expect much better marketing plans, additional revenue opportunities, improved customer services, healthier operational efficiency, competitive benefits, and much more. It is important to not only store data but also process it in order to generate information that is necessary. Every company collects data and uses it; however, to potentially flourish more effectively, a company needs to search relevant data. Every company must carve out direct search-produced data, which can improve their business either directly or indirectly.

Okay, now you have Solr, which is generally referred to as search server, and you are doing searches. Is that what you need? Hold on! This allows a lot more than a simple search. So get ready and hold your breath to take a deep dive into Solr—a scalable, flexible, and enterprise NoSQL search platform!

We will go through the following topics in this lab:

- Introduction to Solr
- Why Solr?
- Solr use cases
- What's new in Solr 7

Introduction to Solr

Solr is one of the most popular enterprise search servers and is widely used across the world. It is written based on Java and uses the Lucene Java search library. Solr is an open source project from **Apache Software Foundation** (**ASF**) and is amazingly fast, scalable, and ideal for searching relevant data. Some of the major Solr users are Netfix, SourceForge, Instagram, CNET, and Flipkart. You can check out more such use cases at https://wiki.apache.org/solr/PublicServers.

Some of the features included are as follows:

- Full-text search
- Faceted search
- Dynamic clustering
- GEO search
- Hit highlighting
- Near-real-time indexing
- Rich document handling
- Geospatial search
- **Structured Query Language** (**SQL**) support
- Textual search
- Rest API
- JSON, XML, PHP, Ruby, Python, XSLT, velocity, and custom Java binary output formats over HTTP
- GUI admin interface

- Replication
- Distributed search
- Caching of queries, documents, and filters
- Auto-suggest
- Streaming
- Many more features

Solr has enabled many such Internet sites, government sites, and Intranet sites too, providing solutions for e-commerce, blogs, science, research, and so on. Solr can index billions of documents/rows via XML, JSON, CSV, or HTTP APIs. It can secure your data with the help of authentication and can be drilled down to role-based authentication. Solr is now an integral part of many big data solutions too.

History of Solr

Doug Cutting created `Lucene` in 2000, which is the core technology behind Solr.

Solr was made in 2004 by Yonik Seeley at CNET Networks for a homegrown project to provide search capability for the CNET Networks website.

Later in 2006, CNET Networks published the Solr source code to ASF. By early 2007, Solr had found its place in some of the top projects. It was then that Solr kept on adding new features to attract customers and contributors.

Solr 1.3 was released in September 2008. It included major performance enhancements and features such as distributed search.

In January 2009, Yonik Seeley, Grant Ingersoll, and Erik Hatcher joined Lucidworks; they are the prime faces of Solr and enterprise search. Lucidworks started providing commercial support and training for Solr.

Solr 1.4 was released in November 2009. Solr had never stopped providing enhancements; 1.4 was no exception, with indexing, searching, faceting, rich document processing, database integration, plugins, and more.

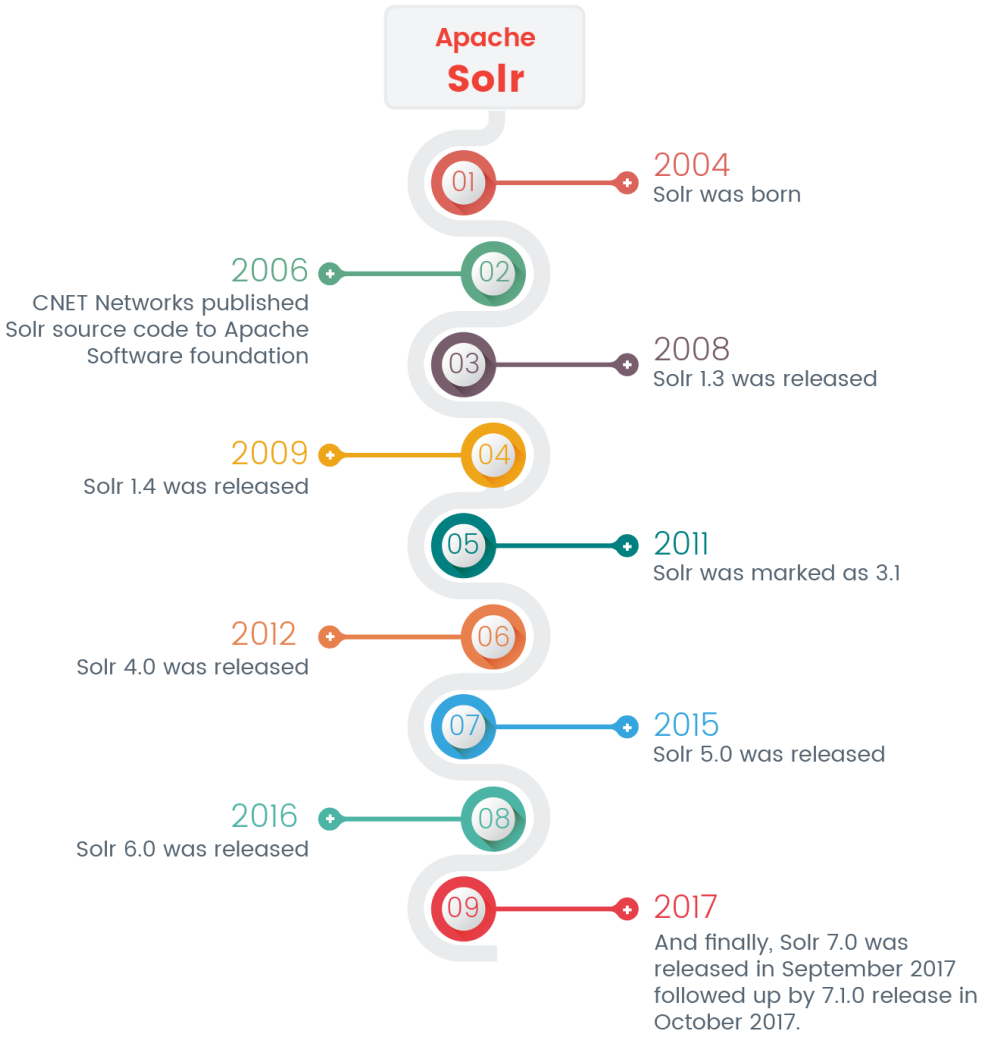
In 2011, Solr versioning was revised to match up with the versions of Lucene. Sometime in 2010, the Lucence and Solr projects were merged; Solr had then became an integral subproject of Lucene. Solr downloads were still available separately; however, it was developed together by the same set of contributors. Solr was then marked as 3.1.

Solr 4.0 was released in October 2012, which introduced the SolrCloud feature. There were a number of follow-ups released over a couple of years in the 4.x line. Solr kept on adding new features, becoming more scalable and further focusing on reliability.

Solr 5.0 was released in February 2015. It was with this release that official support for the WAR bundle package ended. It was packaged as a standalone application. And later, in version 5.3, it also included an authentication and authorization framework.

Solr 6.0 was released in April 2016. It included support for executing parallel SQL queries across SolrCloud. It also included stream expression support and JDBC driver for the SQL interface.

Finally, Solr 7.0 was released in September 2017, followed by 7.1.0 in October 2017, as shown in the following diagram. We will discuss the new features as we move ahead in this lab, in the [*What is new in Solr 7] [section.*]



We have depicted the history of Solr in the preceding image for a much better view and understanding.

So by now, we have a brief understanding of Solr, along with its history. We must also have a good understanding of why we should be using Solr. Let's get the answer to this question too.

Lucene -- the backbone of Solr

Lucene is an open source project that provides text search engine libraries. It is widely adopted for many search engine technologies. It has strong community contributions, which makes it much stronger as a technology backend. Lucene is a simple code library that you can use to write your own code by using the API available for searching, indexing, and much more.

For Lucene, a document consists of a collection of fields; they are name-value pairs consisting of either text or numbers. Lucene can be configured as a text analyzer that tokenizes a field's text to a series of words. It can also do further processing, such as substituting with synonyms or other similar processes. Lucene stores its index on the disk of the server, which consists of indexing for each of the documents. The index is an inverted index that stores the mapping of a field to its relevant document, along with the position of the word from the text of the document. Once you have the index in place, you can search for documents with the input of a query string that is parsed

accordingly to Lucence. Lucene manages to score a value for each of the relevant documents and the ones that are high-scoring documents are displayed. Why choose Solr?

If we already have a relational database, then why should we use Solr? It's simple; if there is a use case that needs you to search, you need a search engine platform like Solr. There are various use cases that we will be discussing further in the lab.

Databases and Solr have their own pros and cons. In one place where we use a database, SQL supports limited wildcard-based text search with some basic normalization, such as matching uppercase to lowercase. It might be a costly query as it does full table scans. Whereas in Solr, a searchable word index is stored in an inverse index, which is much faster than traditional database searches.

Let's look at the following diagram to understand this better:

Solr v.s. relationalDB		
Lucene	Solr	Relational DB
Text Search	Fast and sophisticated	Minimal and slow
Features	Few, targeted to text search	Many
Deployment Complexity	Medium	Medium
Administration Tools	Minimal open source projects	Many open source & commercial
Monitoring Tools	Weak	Very Strong
Scaling Tools	Automated, medium scale	Large scale
Support Availability	Weak	Strong
Schema Flexibility	Must in general rebuild	Changes immediately visible
Indexing Speed	Slow	Faster and adjustable
Query Speed	Text search is fast & predictable	Very dependent on design & use case
Row Addition/Extraction Speed	Slow	Fast
Partial Record Modification	No	Yes
Time to visibility after addition	Slow	Immediate
Access to internal data structures	High	None
Technical knowledge required	Java(minimal),web server deployment, IT	SQL, DB-specific factors, IT
Regular maintenance tasks		

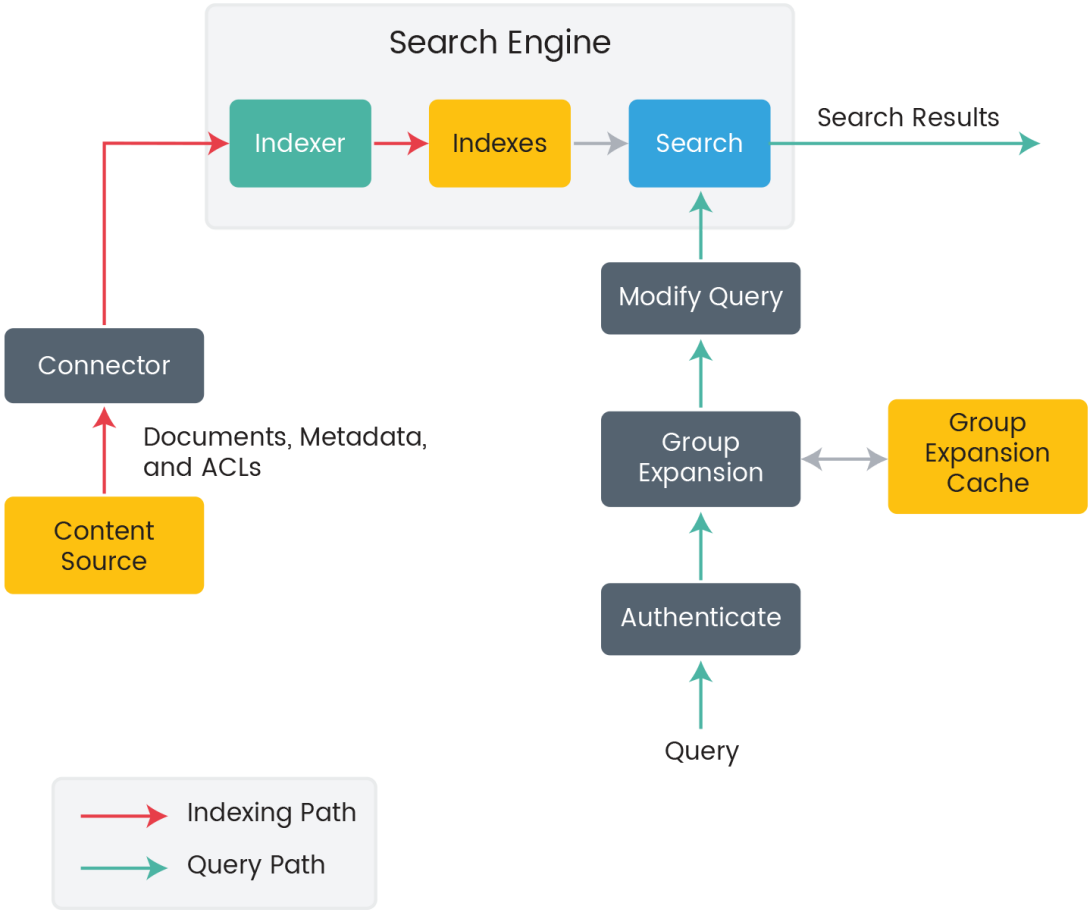
Having an enterprise search engine solution is must for an organization nowadays, it is having a prominent role in the aspect of getting information quickly with the help of searches. Not having such a search engine platform can result in insufficient information, inefficiency of productivity, and additional efforts due to duplication of work. Why?

Just because of not having the right information available quickly, without a search; it is something that we can't even think of. Most such use cases comprise the following key requirements:

1. Data collected should be parsed and indexed. So, parsing and indexing is one of the important requirements of any enterprise search engine platform.
2. A search should provide the required results almost at runtime on the required datasets. Performance and relevance are two more key requirements.
3. The search engine platform should be able to crawl or collect all of the data that it would require to perform the search.
4. Integration of the search engine along with administration, monitoring, log management, and customization is something that we would be expecting.

Solr has been designed to have a powerful and flexible search that can be used by applications; whenever you want to serve data based on search patterns, Solr is the right fit.

Here is a high-level diagram that shows how Solr is integrated with an application:



The majority of popular websites, including many Intranet websites, have integrated search solutions to help users find relevant information quickly. User experience is a key element for any solution that we develop; and searching is one of the major features that cannot be ignored when we talk about user experience.

Benefits of keyword search

One of the basic needs a search engine should support is a keyword search, as that's the primary goal behind the search engine platform. In fact it is the first thing a user will start with. Keyword search is the most common technique used for a search engine and also for end users on our websites. It is a pretty common expectation nowadays to punch in a few keywords and quickly retrieve the relevant results. But what happens in the backend is something we need to take care of to ensure that the user experience doesn't deteriorate. Let's look at a few areas that we must consider in order to provide better outcomes for search engine platforms using Solr:

- Relevant search with quick turnaround
- Auto-correct spelling
- Auto-suggestions
- Synonyms
- Multilingual support
- Phrase handling---an option to search for a specific keyword or all keywords in a phrase provided
- Expanded results if the user wants to view something beyond the top-ranked results

These features can be easily managed by Solr; so our next challenge is to provide relevant results with improved user experience.

Benefits of ranked results

Solr is not limited to finding relevant results for a user's search. Providing the end user with selection of the most relevant results, that are sorted, is important as well. We will be doing this using SQL to find relevant matching pattern results and sorting them into columns in either ascending or descending order. Similarly, Solr also does sorting of the result set retrieved based on the search pattern, with a score that would match the relevancy strength in the dataset.

Ranked results is very important, primarily because the volume of data that search engine platforms have to dig through is huge. If there is no control on ranked results, then the result set would be filled with no relevancy and would have so much data that it wouldn't be feasible to display it either. The other important aspect is user experience. All of us are now used to expecting a search engine to provide relevant results using limited keywords. We are getting restless, aren't we? But we expect a search engine platform to not get annoyed and provide us relevant ranked results with few keywords. Hold on, we are not talking of Google search here! So for users like us, Solr can help address such situations by providing higher rankings based on various criteria: fields, terms, document name, and a few more. The ranking of the dataset can vary based on many factors, but a higher ranking would generally be based on the relevancy of the search pattern. With this, we can also have criteria such as gender; with the rankings of certain documents being at the top.

Solr use cases

Solr is widely accepted and used by big companies such as Netflix, Disney, Instagram, The Guardian, and many more. Let us see with the help of a few use cases the real-life importance that Solr has made on renowned scenarios.

For an extended but incomplete list of use cases and sites that leverage Solr, you can refer to the official web page of Solr at https://wiki.apache.org/solr/PublicServers:

This diagram helps us understand Solr as a solution serving various industries. Though it's not an exhaustive list of industries where Solr has been playing a prominent role in business decisions, Let's discuss a few of the industries.

Social media

LinkedIn, a well known professional social media site, uses Lucene/Solr search. Lucene has a powerful faceting system that allows us to pivot and navigate by user or company attributes abstracted from user profile data. LinkedIn has an excellent feature that is backed up by Solr: its ranking of results by people's relationship with you. This data is not fixed, and being derived by Lucene in real time, it's all based on the arithmetic calculations of the relationships in your connections list.

One more use case is Myspace. Myspace is considered one of the world's largest search sites, with almost 200 million active users and adding up to almost 2.5k new users daily. It is expected to have around 50 million videos and adding around 75,000 daily. Myspace consists of almost 900 billion rows of data and 15 billion friend relationship searches by Lucene, with about 1 terabyte of data added every week.

Science and research

NASA (https://www.nasa.gov/open/nebula.html) uses Solr for its Nebula Cloud Computing Platform. Similarly, **Public Library of Science** (**PLOS**) that is a non-profit publisher of research articles on various subjects. VUFind (https://vufind.org/vufind/) is another powerful open source discovery portal of libraries. It is known to have around 25 million records for a few of its implementations.

Search engine

Having Google using Solr is a milestone for Solr. **Google Search Appliance** (**GSA**), is backed up with Solr. GSA uses many features of Solr: metadata sorting, recommendations, spellcheck, auto-suggest, and more.

Similarly, Open Test Search (http://www.opentestsearch.com/) uses Solr to provide a comparison of a few common search engines.

E-commerce

Flipkart is a leading example of Solr. It has more than 900k users and sees more than 20k searches per second. Flipkart product search has a backbone of 175 million listings, ~250 million documents, and ~5,500 categories. The major challenge was real-time results, ranking, autocompletion, high-update rates, and inverted index. It has become a huge success by using Solr for product searches for its e-commerce business.

Media and entertainment

Netflix uses Solr for the site search feature. Netflix has more than 2 million queries per day for searches and more than 15 million subscribers. It is available in more than 190 countries and supports around 23 languages. The search works based on video title name, genre name, or person name. Features such as autocompletion and ranked results are used by Netflix.

The Guardian, one of the leading newspapers, also uses Solr for its API search platform. There are other users too: MTV, Digg, cnet.com, and many more.

Government

The White House uses Solr for https://www.whitehouse.gov/. It uses features such as search, highlighting, and faceting. Similarly, **Federal Communications Commission** (**FCC**) uses Solr for its website search.

Education

Hathitrust is another wonderful use case of Solr. It has almost a couple of terabytes of index, with more than 10 million books provided online. Solr plays a prominent role in searches through its huge library of books. There are many such examples having similar use cases:

- **Internet Archive**: https://archive.org/
- **National Library of Australia**: http://trove.nla.gov.au/
- **FictFact**: https://www.fictfact.com/
- **Biblio**: https://www.biblio.com/
- **A Norwegian online book store**: https://www.akademika.no/{.ulink}

What's new in Solr 7?

With a major release of Solr, lots of new features have been introduced. Overall, there are 51 new small-to-major features introduced in Solr 7. Along with these features, lots of bug fixes, optimization, and updates have been introduced. Let us go through some of the major changes introduced in Solr 7.

Replication for SolrCloud

Before we understand the new replication methods introduced in Solr 7, Let's go through what was available for replication before Solr 7.

Until Solr 7, Solr had two options for replication purposes:

- Master-slave replication or index replication
- Solr Cloud

In master-slave replication, also known as **index replication**, the master shares a copy of indexed data with one or more slave servers. The master server's job is to index the data that is being added into Solr and share it with the slave servers while all read operations are performed in the slaves.

SolrCloud is a clustered environment of Solr that provides high availability and failover capability so that the content indexed using Solr can be distributed equally among multiple servers for scaling. In SolrCloud, one of the servers act as the leader and the rest of the servers in the cluster work as replica shards. Until Solr 7, in case of any issue on the leader server, any of the replica servers could act as a leader and form the leader-replica cluster. So in that case, data had to be shared with each of the nodes in the cluster, as leader shards and replica shards must remain in sync at any time. Each replica node performed the same operations as the leader. This replication, method available in SolrCloud before Solr 7, was known as NRT replicas.

In Solr 7, two new replication methods have been introduced:

- TLOG replicas
- PULL replicas

TLOG replicas

TLOG replica means transaction log replica. Instead of indexing the data again, a TLOG replica reads the transactions logs of the master or leader shard and replicates the segment or indexed data using a replication handler. In case of failure of the leader shard, one of the TLOG replicas acts as a leader and performs real-time indexing. It also makes a copy of the transaction log. Once the leader shard is available again, it again goes to the replica shard mode and performs only binary replication of segments. Replication done using the TLOG replication method is not as real-time as the one done using NRT replicas.

PULL replicas

A PULL replica pulls the data from the leader shard instead of indexing data locally as in NTR replicas or maintaining the transaction logs as in TLOG replicas. In case of failure of the leader shard, a PULL replica cannot become the new leader shard. For that, we may have to use either TLOG or NRT only. PULL replicas provide faster data replication from leader shards to replica shards.

Schemaless improvements

Solr has improved its schemaless mode functionality, the way it now detects data for indexing of an incoming field would be text based. By default, it will now be indexed as `text_general` for incoming fields, which can be modified if required. The name of the field will be the one defined in the document. A copy field rule will now be added in the schema when a collection is created if config set is not defined. It is now schemaless, which would insert the first 256 characters from the text field in a new strings field. It would be named as `<name>_str`.

The relevant schemaless behavior can be customized to remove a copy field rule or to update the number of characters added into the strings field or type of field used.

Copy field rules can impact the index size as well as slow down the indexing process. It is recommended to use the copy field rule when it is required. If there is no need to do a sort or facet on a field, you should ideally disable the copy field rule that is generated automatically.

The field creation rule can be disabled via the `update.autoCreateFields` property. You can also use the configuration API with the following command to disable it:

```
curl http://hostname:8983/solr/collection/config -d '{"set-user-property":
{"update.autoCreateFields":"false"}}'
```

Autoscaling

As termed in the documentation of http://lucene.apache.org{.ulink}, the goal of autoscaling is to make SolrCloud cluster management easier by providing a way for changes to the cluster to be more automatic and more intelligent.

So in Solr 7, there are some APIs that monitor some predefined preferences and policies. If any of the rules provided in the policies are violated, Solr changes its configuration automatically as defined in the preferences. With the updated autoscaling feature, we can now have Solr spin up new replicas depending on the monitoring metrics, such as disk space.

Default numeric types

`Trie*` -based numeric files are now replaced by `*PointField` from Solr 7 onwards. Going forward, from Solr 8, all `*PointField` types will be removed. You need to work towards moving from `*PointFields` to the new `Trie*` fields for your schema. After changing to the new `*Pointfields` type, data will need to be reindexed in Solr.

Spatial fields

Here is the list of spatial fields that have been deprecated:

- `SpatialVectorFieldType`
- `SpatialTermQueryPrefixTreeFieldType`
- `LatLonType`
- `GeoHashField`

The following is the list of spatial fields that can be used moving forward:

- `SpatialRecursivePrefixTreeField`
- `RptWithGeometrySpatialField`
- `LatLonPointSpatialField`

SolrJ

Here are the changes made in SolrJ:

- `HttpClientBuilderPlugin` is replaced with `HttpClientInterceptorPlugin` and would work with a `SolrHttpClientBuilder` rather than `HttpClientConfigurer` that was the case earlier.
- `HttpClient` instances configuration can be done now with help of `SolrHttpClientBuilder` rather than the earlier `HttpClientConfigurer` with the help of `HttpClientUtil`.
- `SOLR_AUTHENTICATION_CLIENT_BUILDER` is now being used in variable instead of `SOLR_AUTHENTICATION_CLIENT_CONFIGURER` in environment variable.
- `HttpSolrClient#setMaxTotalConnections` along with `HttpSolrClient#setDefaultMaxConnectionsPerHost` has now been removed. By default, these parameters are now set on the higher side and can be changed with the help of parameters when an `HttpClient` instance is created.

JMX and MBeans

Here are the changes made in **Java Management Extensions** (**JMX**) and MBeans:

- We notice there is now a hierarchical format for names used in metrics in MBeans attributes. For reference we can have look at `/admin/plugins` and `/admin/mbeans`. And the UI plugins tab is now using a similar approach as now all the APIs fetch data from a metrics API. The earlier approach of having a flat JMX view has been removed.
- `<metric><reporter>` has now replaced `<jmx>` elements in `solrconfig.xml`. And `<metric><reporter>` needs to be defined in the `solr.xml` configuration file. Default instances of `SolrJmxReporter` supports automatically limited backward compatibility when a local MBean server is discovered. If we want to enable a local MBean server we can use `ENABLE_REMOTE_JMX_OPTS` in `solr.sh` configuration file or via system properties that uses `-Dcom.sun.management.jmxremote`. With default instance all registries are exported using Solr metrics.
- If we want to disable the behavior of `SolrJmxReporter` we can do it by using `SolrJmxReporter` configuration with a `Boolean` argument set to `false`. Backward compatibility support might be removed from Solr 8 for `SolrJmxReporter`.

Other changes

Apart from these changes, there are many other features and improvements that have been made in Solr 7:

- In Solr 7 the default response type is set to `JSON` that was previously in XML format. If you want a response in `XML` then you will need to defined `wt=xml` in the request parameter.
- Default value for the `legacyCloud` parameter is set to `false`. That means if an entry is not found for the replica in `state.json`, it will not be registered in the cluster shard.
- By default, the new incoming field will be indexed as `text_general`. The name of the field will be the same as defined in the incoming document.
- The `_default` config set is introduced to replace `data_driven_configset` and `basic_configset`. So while creating a new collection if no configuration value is defined, it will use `_default` configuration. In case of SolrCloud, ZooKeeper will use `_default` configuration if no configuration parameter is defined. While in standalone mode, `instanceDir` will be created using the `_default` configuration parameter.
- New configuration set is defined for the SolrClient. So now configuration of socket timeout or connect timeouts are not dependent on `HttpClient` and can be defined specifically for SolrClient.
- In SolrJ, `HttpSolrClient#setAllowCompression` that was earlier used to define enabling compression has been removed. Now this parameter must be enabled from the `Constructor` parameter only.
- New V2 **Application Program Interface** (**API**) is available at `/api/` as a preferred method and to leverage old API `/solr/` continues to be available.
- The standard query parser now has the default `sow=false` which means that text fields will not split on whitespace before handing text to the analyzer. It will help analyzer to match synonyms of multi-words. Summary

---

This was an interesting lab with important content to learn, right? In this lab, we started with an introduction to Solr. We went through the impressive history of Solr and its backbone, Lucene. We learned and understood the characteristics and why we use Solr. Besides, we saw a wonderful list of use cases around the globe that have adopted Solr to serve many of the key features of the solution. In the later section of the lab, we learned about the exciting new features of Solr 7 that briefly covered replication, SolrJ, schemaless improvements, spatial fields, and more.

Moving on to the next lab, we will learn how to get started with Solr. The lab will focus on Solr installation, understanding various files and the folder structure, along with loading sample data into Solr. The lab will explain

how we can browse the Solr interface and how to use the Solr admin interface.