Getting Started with Solr

Table of Contents

Solr is the popular open source search solution.

It is highly reliable, scalable and fault tolerant, providing distributed indexing, index replication, load-balanced querying, automated failover and recovery, centralized configuration and more.

Solr can index content from many sources and has integration points for Apache Tika to index rich text documents (Office documents, PDFs, etc.), JSON files, CSV files and Solr-specific XML.

In addition, Lucidworks has provided a set integration points to index content from HDFS, with a Pig script, via a Flume Sink, a Storm bolt, and from a Hive table.

If you have never used Solr before, these sections describe concepts and resources to help you get the most out of Solr from the start. If you already familiar with Solr, much of this content may already be familiar to you.

First we'll describe concepts critical to SolrCloud and using Solr in a cluster. Next, we'll walk through configuring indexes, adding documents and querying the system.

Cores, Collections and Clusters

Generally speaking, if you use Solr in *standalone mode*, you have a single core for each index. You can have multiple cores, but they would all be separate indexes.

If you use Solr in *SolrCloud mode*, which is how this documentation suggests you use Solr with Hadoop, you would have a core on each node of your cluster, and together those cores make up a collection. You can have multiple collections, for separate indexes.

For more information about the basics of SolrCloud, see How SolrCloud Works in the Apache Solr Reference Guide.

Terminology: Cores, Collections & Nodes

There are several terms that are used to describe parts of a SolrCloud implementation, and it's helpful to try to understand them early:

Core

A single Solr instance, which represents a single Solr index. A core has a different set of configuration files and schema definitions than other cores.

Collection

A group of cores that together form a single logical index. A collection has a different set of configuration files and schema definitions than other collections.

Shard

A logical section of a single collection.

Node

A Java Virtual Machine instance running Solr, commonly known as a server. Multiple cores can run on a node if you wish.

While a single core includes a single index, an index can also be distributed across multiple nodes. In this case, the part of the index on any single node is a shard. Each shard is hosted in a core.

Clusters and SolrCloud

A cluster is a group of nodes managed together by ZooKeeper. When a new core is added to a cluster, it registers with ZooKeeper, which then keeps track of the status of that core.

Each node is either a *leader* or a *replica*. A leader is similar to a "master" node, and is responsible for making sure replicas are up to date with the same information as the leader. A replica is similar to a "worker" or "slave" node, which contains a copy (a "replica") of the index and can serve queries to the index. This provides a level of failover and redundancy.

As updates are made to the index, they are distributed through the cluster. Queries are also distributed, and load balancing is supplied automatically from ZooKeeper.

When planning your cluster, it’s often best to *overshard*, which means to start with more shards per node than you expect to have in production and then move the shards to new hardware as they grow too large to share a single node of your cluster. This strategy allows you to grow without needing to consider splitting the index to new shards. While Solr allows you to split shards, if you start with a higher number of shards, you get the benefits of increased parallelism during your implementation phases.

Clusters can be resized if necessary. A Collections API allows using HTTP requests to modify a collection. More details on this are available in the Apache Solr Reference Guide section Collections API.

Setting Up Your Index

Choosing Single or Multiple Indexes

As mentioned above, you can have separate indexes that co-exist in the same installation of Solr (it does not matter if these are cores or collections). Why would you want separate indexes? Generally, these scenarios usually drive users to create different indexes:

- You have documents that are so different that it does not make sense to search them together, such as a DVD database and a set of log files.
- Your documents have different access restrictions for who is able to query or view them in results.
- You want to provide different query interfaces to different users, such as customers in the US and customers in Europe.

When you use Solr in SolrCloud mode, you achieve separation of indexes by creating different collections. Automatically, you will be working with separate indexes.

Fields and Your Index Schema

In Solr, all content is represented as a *document*. A document is set of data that describes something. A document in this context is not the same as a file. A file may consist of several documents. For example, if you have a set of log files, you may want to treat each log entry in each file as individual documents. This would allow you to do queries that pinpoint specific events without having to read the whole file, or use your log file to monitor trends in activity.

Each document is made up of *fields*. Using a log file as an example, each event in the log may include the date and time, the request made to the system, how long the system took to respond to the request, and so on. Each of these fields can be different data types - dates, text, strings, numbers, etc.

Further, Solr includes a way to transform content as it's being added to the index. This allows you to correct for the same word in different cases (lower vs. upper). This process is called *analysis* and you can perform analysis on documents during indexing and also on queries being submitted by users.

In Solr, the fields that comprise a document and all of the rules for processing content found in the fields is defined in `schema.xml`. This file is found on the filesystem in the `confDir` for your core or collection.

Schemaless vs. Planning Your Schema

While Solr includes a schema of fields by default, it can also guess at the proper field names and field type definitions during indexing. This is called *Schemaless Mode*. There are three main features of running Solr in schemaless mode:

1. Managed schema. The managed schema feature in Solr provides APIs to programmatically update your schema instead of manually editing the `schema.xml` file.
2. Field type guessing. During indexing, the type of content in the document is guessed, using parsers. Several types of fields are supported:

  - Boolean
  - Integer
  - Long
  - Float
  - Double
  - Date

3. Automatic field addition. When new fields are found, they will be added to the schema.

These are powerful and flexible features, and can save a lot of time when just starting with Solr.

While schemaless provides an easy way to get started, the analysis and planning that are required to design your schema before indexing your documents gives you a higher level of control over the indexing process. This allows you to create a more consistent experience for users when they submit queries.

A hybrid approach may be sufficient in some use cases. It’s always worth taking the time to design for specific fields that are most important but you can decide to let Solr make educated guesses for fields that are not as important.

If you created your collection with the `data_driven_schema_configs` (as in the example in the installation section of this documentation), you are by default using schemaless mode.

For more information about managing your schema, see Schemaless Mode in the Apache Solr Reference Guide.

More about Managed Schema

The managed schema feature can be used without using schemaless mode's other features of automatic field guessing and field addition.

The cornerstone of this feature is an API to manage schema elements such as fields and field types. In fact, when using this feature, schema modifications can *only* be made with the API. This approach has two main benefits:

1. If you implement Solr's security features, you can control who can make schema modifications and what kind of modifications they are allowed. This way you can provide access to schema editing (or simply reading) to only parts of the schema without providing access to the server's filesystem.

   More details on Solr's security features are available in the section Securing Solr in the Apache Solr Reference Guide.
2. In SolrCloud mode, the Schema API allows you to make modifications to the schema without having to interact with ZooKeeper to download and upload the file before and after editing. For new users in particular, getting config files from ZooKeeper is frequently a troublesome process; the API allows you to avoid it.

More details on this feature are available in the Apache Solr Reference Guide in these sections:

- Managed Schema Definition in SolrConfig explains how to set up the managed schema feature.
- Schema API describes the API calls available.

Adding Documents

Indexing Documents

Adding content to Solr is called *indexing*. There are several supported approaches to indexing your content:

Upload XML, JSON or CSV

If you have your content in XML/XSLT, JSON or CSV already (or can easily get it into those formats), you can upload files directly to Solr for indexing.

This utilizes a feature of Solr called an *index request handler*, or more simply an *index handler* which knows how to process content in these formats.

More information is available from the Apache Solr Reference Guide in the section Uploading Data with Index Handlers.

Upload using Apache Tika

If your content is in a format supported by Apache Tika, such as PDF files or Word documents (among many other formats), you can use a request handler called the `ExtractingRequestHandler`, also known as Solr Cell (for Solr Content Extraction Library).

The ExtractingRequestHandler parses incoming files with Apache Tika and extracts fields to be indexed by Solr.

In practice, this works very similarly to the index handlers, but when making the request you use a different request format and must structure the request properly to use Tika instead of an index handler.

More information is available from the Apache Solr Reference Guide in the section Uploading Data with Solr Cell using Apache Tika.

Import from a Database

Solr includes a plugin called the DataImportHandler. This plugin, with the help of a JDBC driver, can connect to a database to import rows and use column names as field names. If your content is in Oracle, MySQL, Postgres or any other relational database, the DataImportHandler may be a good way to index that content to Solr.

The DataImportHandler supports more than only imports from JDBC databases. It can also import emails, RSS feeds, XML data, plain text files, and more.

More information is available from the Solr Reference Guide in the section Uploading Structured Data Store Data with the Data Import Handler.

Using the Post Tool

Solr includes a tool that is a Unix-based shell script to "post" documents to Solr. This tool includes basic web crawling, in addition to automatically understanding when to use an index handler or Solr Cell, both described above.

More information about using the Post Tool is available from the Solr Reference Guide in the section Post Tool.

Querying the Index

About Queries in Solr

The primary way to ask questions of Solr is to send an HTTP request. Solr also has support for using clients.

When you send a query, Solr processes it with a *query request handler* or, more simply, a *query handler*. This is similar in concept to an index handler, but it is designed to return documents from the index.

You can define several query handlers to handle different types of requests, and set defaults for all of the available options, such as the fields to query by default, the fields to return in the response, the number of documents to return, whether (or how) the response should include facets, highlighted query terms, and many, many other features.

More information on request handlers is available in the Solr Reference Guide section RequestHandlers and SearchComponents in SolrConfig.

Query Format

Formatting queries in Solr is quite simple. As an HTTP request, simply send the request to an endpoint that corresponds to the name of the request handler you've configured.

By default, one is provided out of the box, named `/select`. A query to `/select` is as simple as this request:

```
http://localhost:8983/solr/select?q=*:*
```

The `*:*` query term used in this example is Solr shorthand for requesting every document in the index. Essentially, using wildcards, that request tells Solr to return every field for every indexed term in the index.

The response can be returned in several possible formats, whichever is more convenient for your client application. The format to return is usally sent as part of the request, with the `wt` parameter, such as `wt=json`.

The Solr Admin UI includes a query interface that allows you to use a form to play with query options.

Response Format

Solr can send query responses in several formats but the most frequently used are JSON and XML.

The format will include a `responseHeader` which can include the parameters of the query. Depending on the query parameters, the response will additionally include:

`response`
: This will include subsections (`docs`) for the documents that match the query terms.

`facet-counts`
: If facets have been requested, the facet calculations.

`highlighting`
: If highlighting has been requested, the document snippets which include coded sections highlighting the user’s search terms.

`debug`
: If requested, Solr can include debugging information as part of the response, which includes how Solr parsed the user’s query, the scoring data, and the time to process both the query and the results.