Lab 2. Getting Started

Information is the oil of the 21st Century, and analytics is the combustion engine.

Apache Solr is one tool that has some common features such as full-text search, facets-based search, clustering on demand, the ability to select highlighters, indexing on demand, database integration, geospatial searches, NoSQL features, fault tolerance, scaling on demand, and the ability to handle all sorts of documents (such as PDF, docx, ppt, and more). Solr finds its intense use in enterprise searches and huge analytics. Due to such powerful features, major players in the world such as Instagram, Netflix, NASA, and eBay are using Solr.

Solr runs as a standalone server with the option of having clusters as per demand. Internally, it makes use of the Lucene Java search library and has full support of REST like HTTP/XML and JSON APIs. This makes it possible to use most of the available programming languages even without Java coding. You feed Solr data over HTTP and Solr responds with an output in formats of JSON, CSV, and binary. Apache Lucene and Apache Solr were merged in 2010 and they are collectively termed Lucene/Solr or Solr/Lucene. This lab focuses mainly on the following points:

- Solr installation
- Understanding various files and the folder structure
- Running and configuring Solr
- Loading sample data
- Understanding the browser interface
- Using the Solr admin interface

Solr installation

Let's get up and running with Solr. At the time of writing of this book, the latest stable version of Solr was 8.6.0. This course focuses all its aspects on Apache Solr 8.6.0.

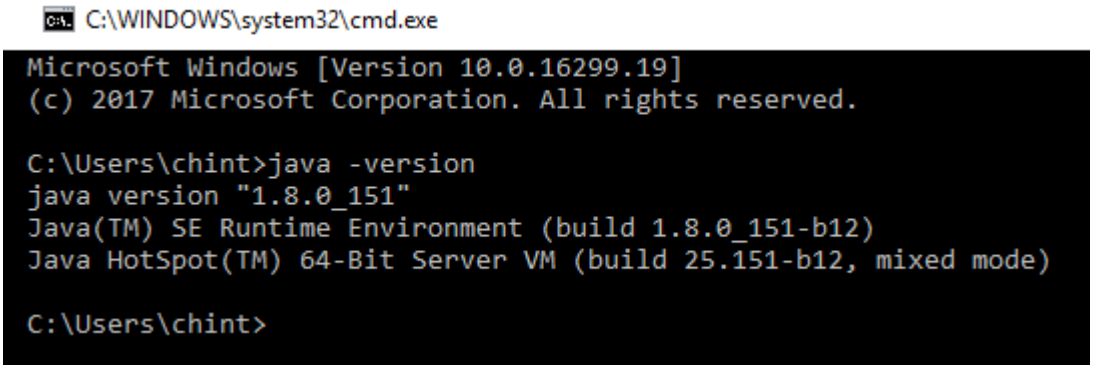
Getting up and running with Apache Solr requires the following prerequisites:

- Java 8 (mandatory)
- curl (optional, recommended)

So, you will necessarily require JRE version 1.8 or higher for Solr to run. Go ahead! Open Command Prompt, type the following command, and check your Java version:

```
java -version
```

It should show the following output:

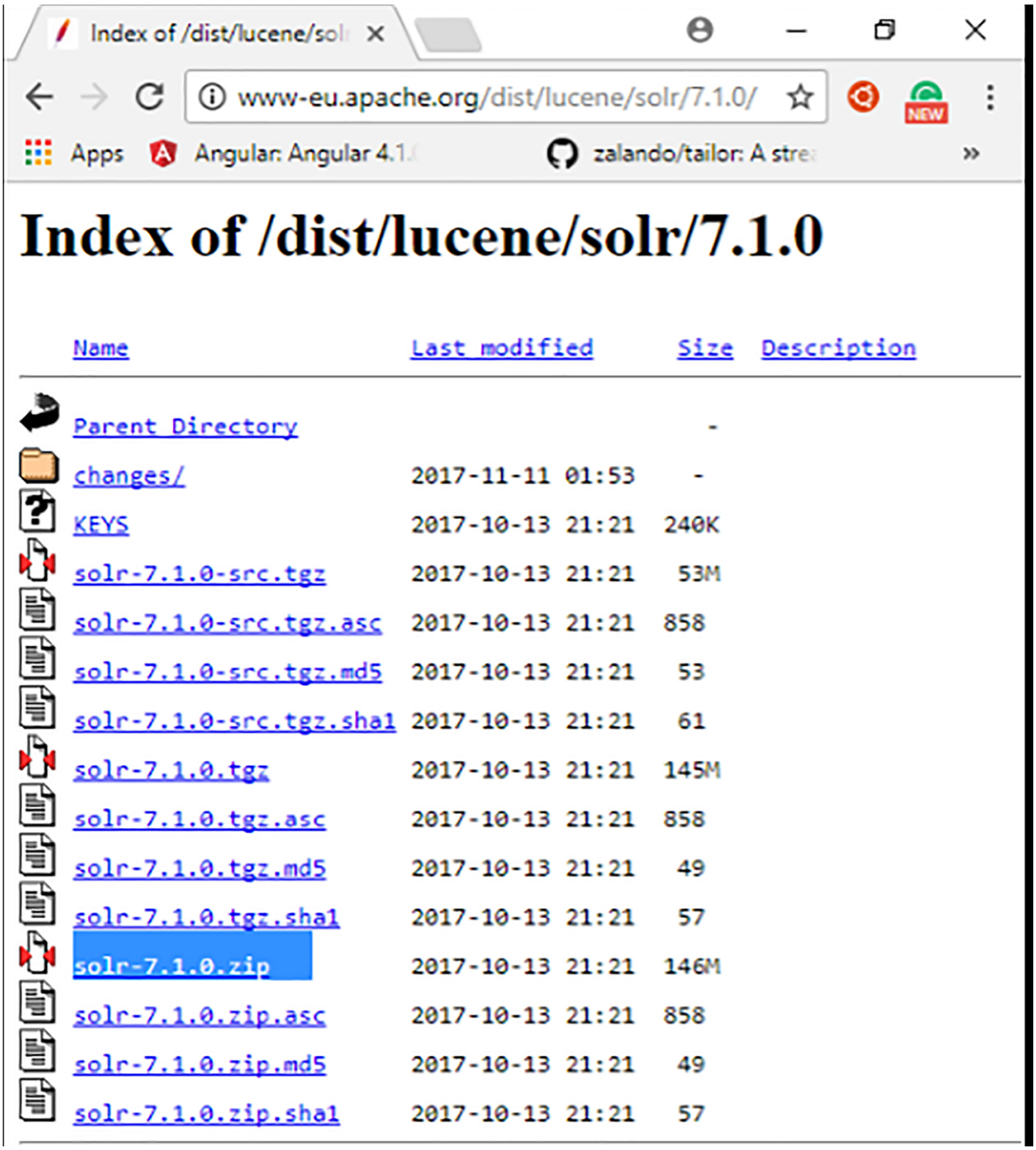


If your output is something different, it means Java is not properly installed and it needs to be installed properly. Also, the `PATH` variable should point to JRE 1.8. You can download Java from http://www.oracle.com/technetwork/java/javase/downloads/index.html if needed.

Curl utilities are not mandatory but will be helpful when we go deeper.

The next step is to download Apache Solr. An important point to keep in mind is that Apache Solr can also run as a standalone unit and is operating system independent. It would be the same across the operating systems; the difference would be in the way we reach there. To install Solr, follow along steps:

1. Go to http://lucene.apache.org/solr/.
2. Click on **`DOWNLOAD`** .
3. You will be redirected to http://www.apache.org/dyn/closer.lua/lucene/solr/8.6.0.
4. Select the mirror site, which will open up the following page:



5. If you are on Windows, download the `solr-8.6.0.zip` file and extract it to a location of your choice.
6. If you are on Linux/Unix or macOS, download the `.tgz` file and extract it to the local home directory:

NOTE

Solr has been already installed "/opt/solr", their is no need to download following solr setup.

```
wget https://downloads.apache.org/lucene/solr/8.6.0/solr-8.6.0.tgz

cd ~/

tar -zxvf solr-8.6.0.tgz
```

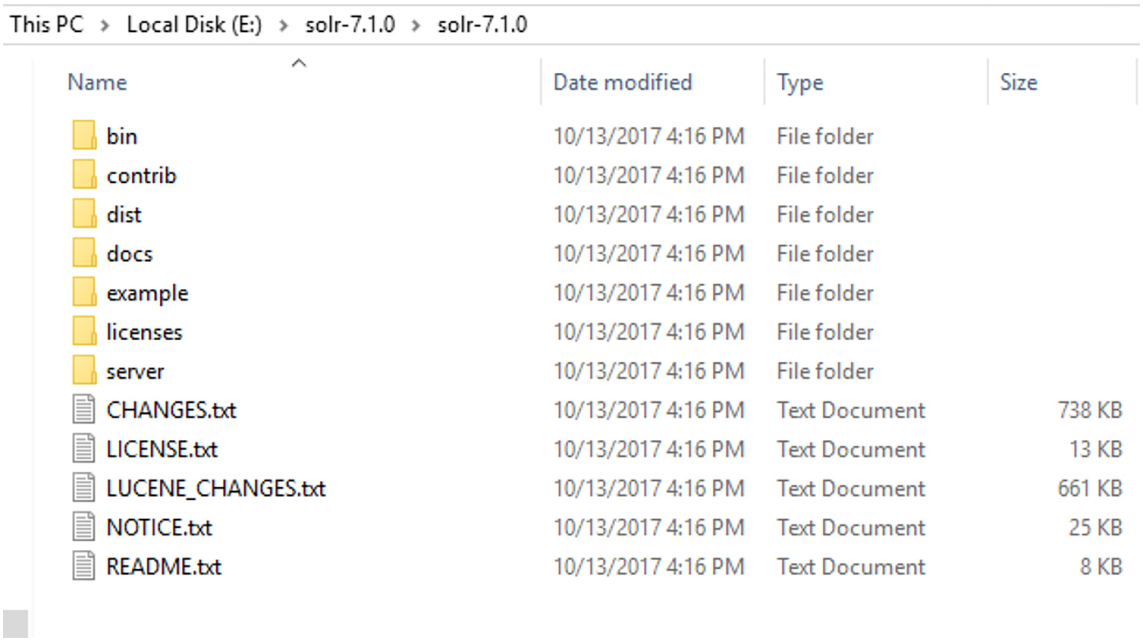
The same command should work for macOS also, provided the `wget` utility is installed. If not, you can use another alternative.

7. For macOS, if you don't have `wget` installed, you can use the native `brew` command:

```
brew install solr

solr start
```

Irrespective of the way you install Solr for your OS, the extracted location should be something like this, with the folder structure as follows:



Solr User Solr

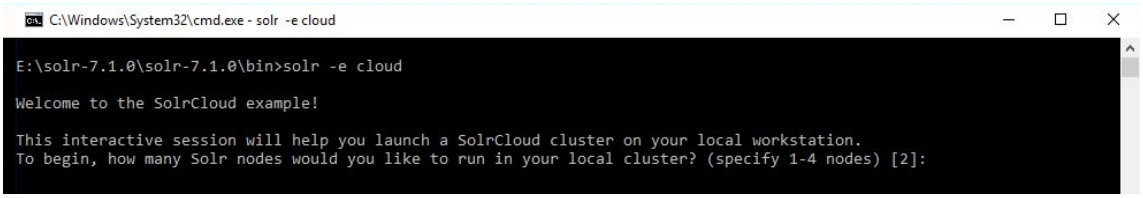
Make sure to switch to **solr** user by running following command or user `-force` flag to start solr service with root user:

```
su solr
```

Now that we have Solr installed, the next step is to get it up and running. Bin folder has been already added to environment path, which contains the Solr startup script. Open the terminal and hit the following command, which

will open up an interactive prompt screen, as shown in the following screenshot:

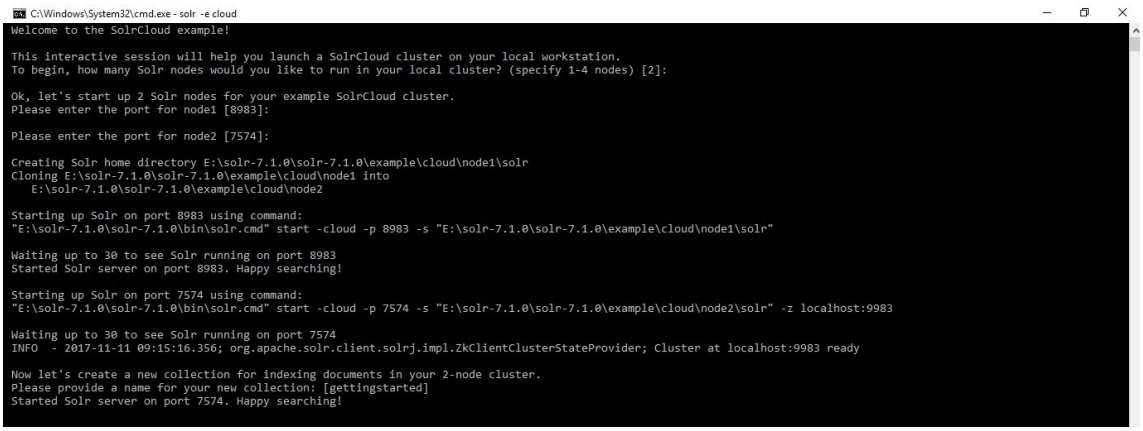
```
solr -e cloud
```



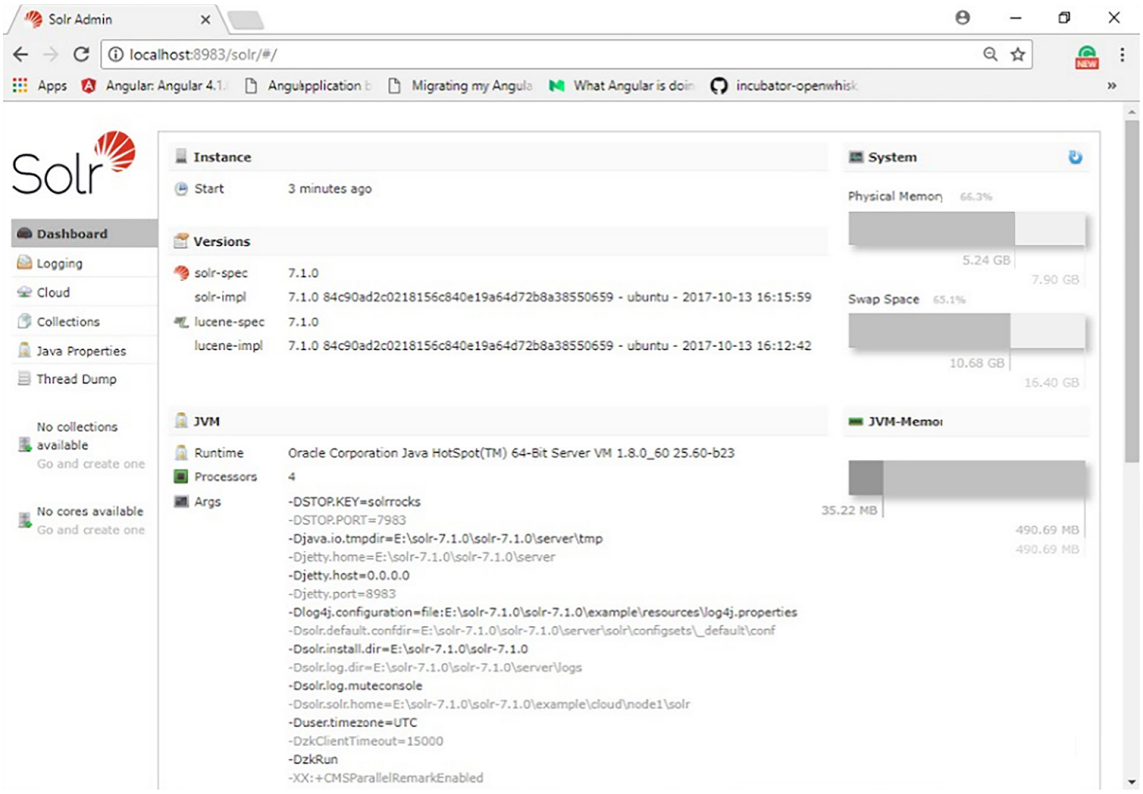
Let's look at the following steps:

1. The very first question it asks us is the number of nodes we want to run in parallel for Solr. Let's keep it at `2` since `2` is the default. Just press [*Enter*] without typing anything.
2. Next, we need to specify the port on which Solr will run. The default ports are `8983` and `7574`. Change the port if they are already occupied, or simply press [*Enter*] to go ahead with the default ports.
3. Next, it will ask us to provide the name of a collection in which we want to add data. Let's go with the default one.
4. The `bin/solr` and `bin/solr.cmd` contains Solr preconfigurations. When we start Solr, it asks us whether we want to change those configurations. We can simply press [*Enter*] to keep the preconfigurations as default, or we can type in to define our own configurations.

If everything goes well, you should see the following screen:



You can now navigate either to `http://localhost:8983/solr/#/` or `http://localhost:7574/solr/#/` as you have started Solr in cluster mode. You should be able to see the following admin interface:



What did we just do? While running this command, we started Solr in cloud mode. When we specified the number of hosts, port number, and collection name, it created an initial configuration set. Solr comes with an example configuration set, which we can use with cloud mode. It can be found at: `$SOLR_HOME/server/solr/configsets`.

If you navigate to that folder, you will be able to see the `_default` and `sample_techproducts_configs` config sets.

However, cloud mode is not the only mode Solr comes with. Solr comes with tailor made configuration sets, which we can use if they fit our purpose. The following config sets are available in Solr 8.6.0 and they can be viewed at `$SOLR_HOME/example`. If you navigate to that folder, you will be able to find config sets for an instance of `cloud`, `example-DIH`, `films`, and `techproducts`.

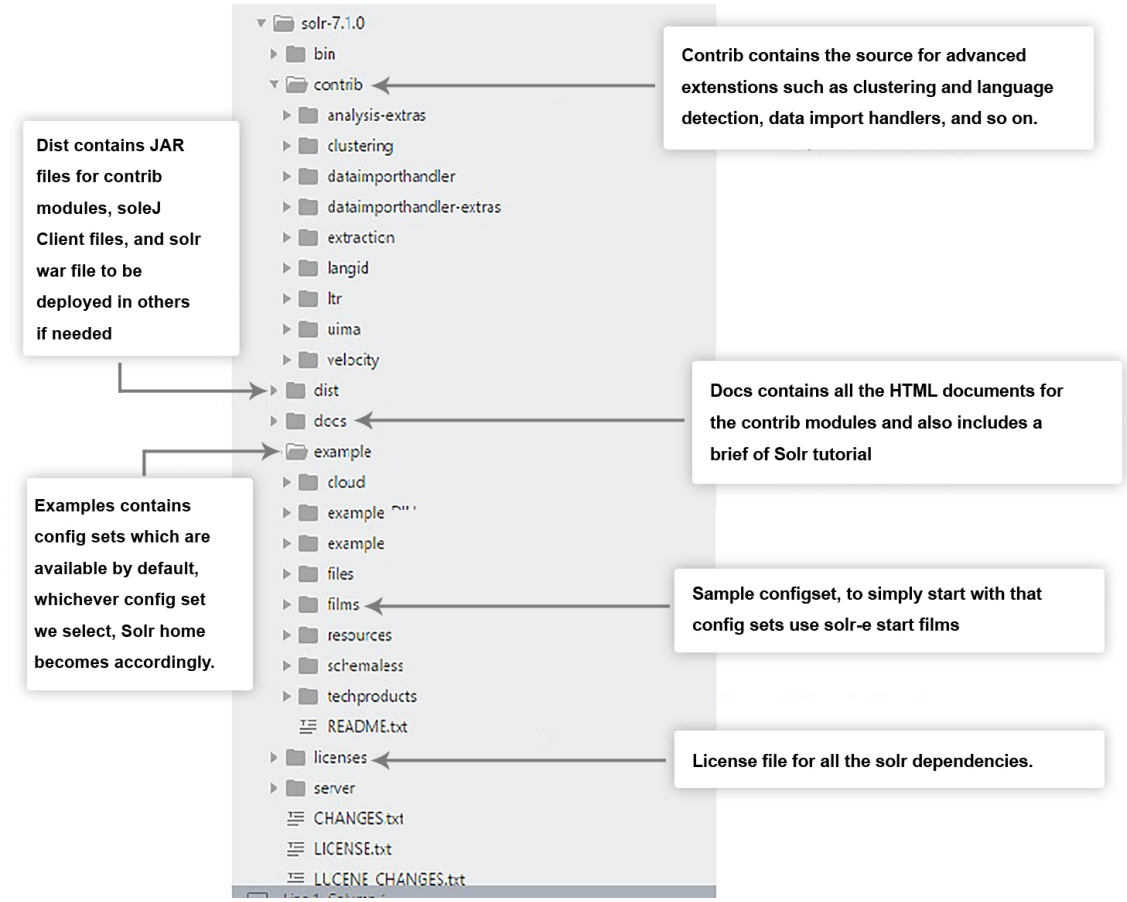
To start Solr with any of the example config sets, you have to navigate to the `bin` directory and type the following command in the terminal:

```
solr start -e <name_of_example_config_set>
```

Now that we are up and running with Solr, Let's look at the folder structure to get a deeper idea of Solr.

Understanding various files and the folder structure

Let's now understand the constituents of Solr, its directory structure, and the significance of each folder and the configurations involved. This diagram shows the parent-level directory of Solr, along with an explanation of each folder:



Let's look at the major folders and files we will be dealing with.

bin

The bin folder has all the scripts primarily needed to get up and running with Solr. Mainly, we will be using Solr and post scripts for our day-to-day purposes. It is also the location to place the replication scripts for more advanced setup, if needed. The `bin` folder is home to the following utilities: it contains the `solr.in.sh` and `solr.in.cmd` files, from where you can provide input parameters to Solr. It has the `install_solr_service.sh` script and the `oom_solr.sh` and `init.d` folders, if you want to run it as a service on any Linux/Unix service.

Since we will be using Solr scripts very often, it is advisable to add `$SOLR_HOME/bin` (for example, `E:\solr-8.6.0\solr-8.6.0\bin`) in the path environment variable so that we can use it anywhere.

Solr script

The Solr script in this folder is used for various utilities for managing Solr. You can pass various parameters to Solr from here. Now we will see a list of some of the things you can do with Solr script. Go to the `bin` folder and execute the commands in the terminal to check the output. Based on the operating system, use the `.cmd` or `.sh` version accordingly:

- If you are on Windows, use the following command:

```
bin\solr.cmd start
```

- If you are on Linux, use this command:

```
bin/solr.sh start
```

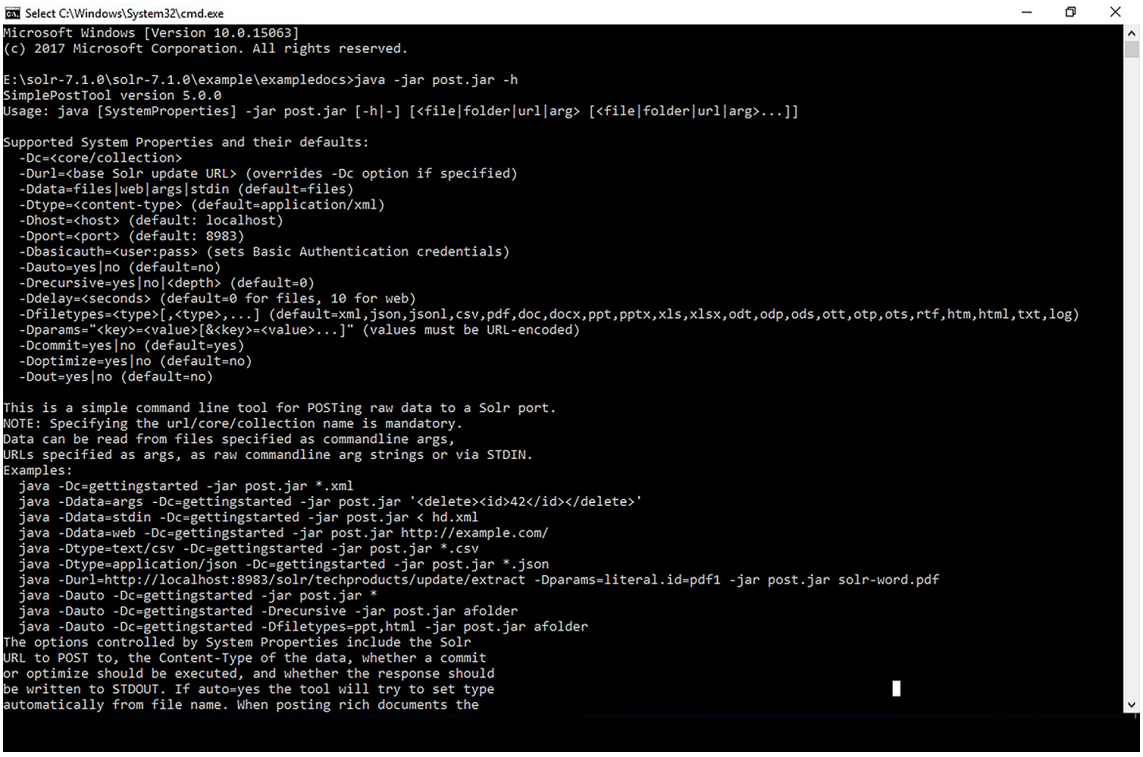
Post script

Solr, by default, includes a post script for indexing various kinds of documents directly to the Solr server. This utility is very helpful when you have huge files present and you want to end up writing a program that reads each line and then sends it to the Solr server via HTTP end points.

Note

To run this utility on Windows, please install Cygwin. Otherwise, you will have to navigate to `$solr_home\example\exampledocs\post.jar`.

The following screenshot helps us to understand how we can execute a post script from the command line:



This table consists of some example commands used for the same purpose:

Command	Description
`post -c chintan *.xml`	This adds all files with extension `.xml` to the core or collection named chintan
`post -c chintan -d '<delete><id>42</id></delete>'`	Deletes a document from the chintan collection or core that has ID `42`
`post -c chintan *.csv`	Indexes all CSV files with auto field mappings on
`post -c chintan *.json`	Indexed all JSON files
`post -c chintan *.pdf`	Indexes all PDF files
`post -c chintan abc/`	Indexes all files inside the `abc` folder
`post -c chintan -filetypes ppt,html abc/`	Indexes only PPT and HTML files inside the `abc` folder

contrib

The `contrib` folder is one where you will be able to see all of Solr's contribution modules. All the extensions to Solr, which do advanced things beyond core Solr, can be found here.

Note that the runnable Java files of all of these extensions can be found in the `dist` folder; here it's just the source and the `README.md` files.

Some of the useful extensions are as follows.

DataImportHandler

`DataImportHandler` is an import tool for Solr that helps in importing data from databases, XML files, and HTTP data sources very smoothly and easily. The data sources for importing go beyond relational databases and cover filesystems, websites, emails, FTP servers, NoSQL databases, LDAP, and so on. You can set default locales, time zones, or charsets via this extension. You can find two JAR files of this extension in the `dist` folder: `solr-dataimporthandler-8.6.0.jar` and `solr-dataimporthandler-extras-8.6.0.jar`.

ContentExtractionLibrary

This contrib module provides a way to extract and index content contained in rich documents, such as Word, Excel, PDF, and so on. This module uses Apache Tika (a toolkit that extracts metadata and text from over a thousand different files). This contrib module provides automatic MIME type detection so that it can use that based on the file provided.

LanguageIdentifier

This module is meant to be used while indexing documents with a multitude of languages. The implementation of this document is such that it creates an `UpdateProcessor`, which can be placed in `UpdateChain`. It identifies the language and tags that document with that `languageId`. For example, if the input is `surname` and it detects English as the language, then it will be `surname_en`. Language detection can be per field or for the whole document.

Clustering

This module is used when we want to add third-party implementations. At the time of writing this course, it provides clustering support for search results using the Carrot2 project (https://project.carrot2.org/).

VelocityIntegration

This contrib helps us to integrate Solr with Apache Velocity. It gives us a writer that, behind the scenes, uses the Apache Velocity template engine on the GUI to render Solr responses.

dist and docs

The `dist` folder contains all the distributions that can be used as deployment artifacts in other servers. It contains the main Solr file, which is `solr-core.8.6.0.jar`. This folder also contains JAR files of all the contribs we discussed earlier. You can deploy these artifacts to any application server as per your needs.

The `docs` folder contains online documentation for all the help needed in Solr. It has Wiki Docs, new changes in the current version, minimum system requirements, tutorials, Lucene documentation, and Java API docs for all the contrib modules.

example

This directory contains all Solr examples and config sets that are provided by default. Each example is self-contained in a separate directory. It contains a fully self-contained Solr installation. It has a sample configuration, some documents, and ZooKeeper data.

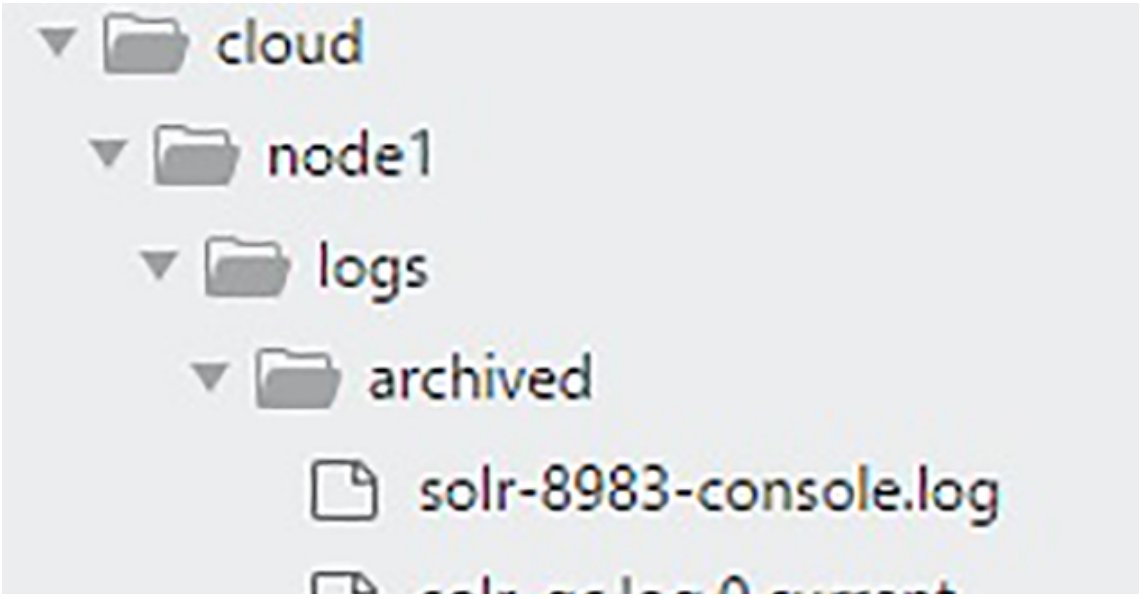
Solr provides the following sample data out of the box:

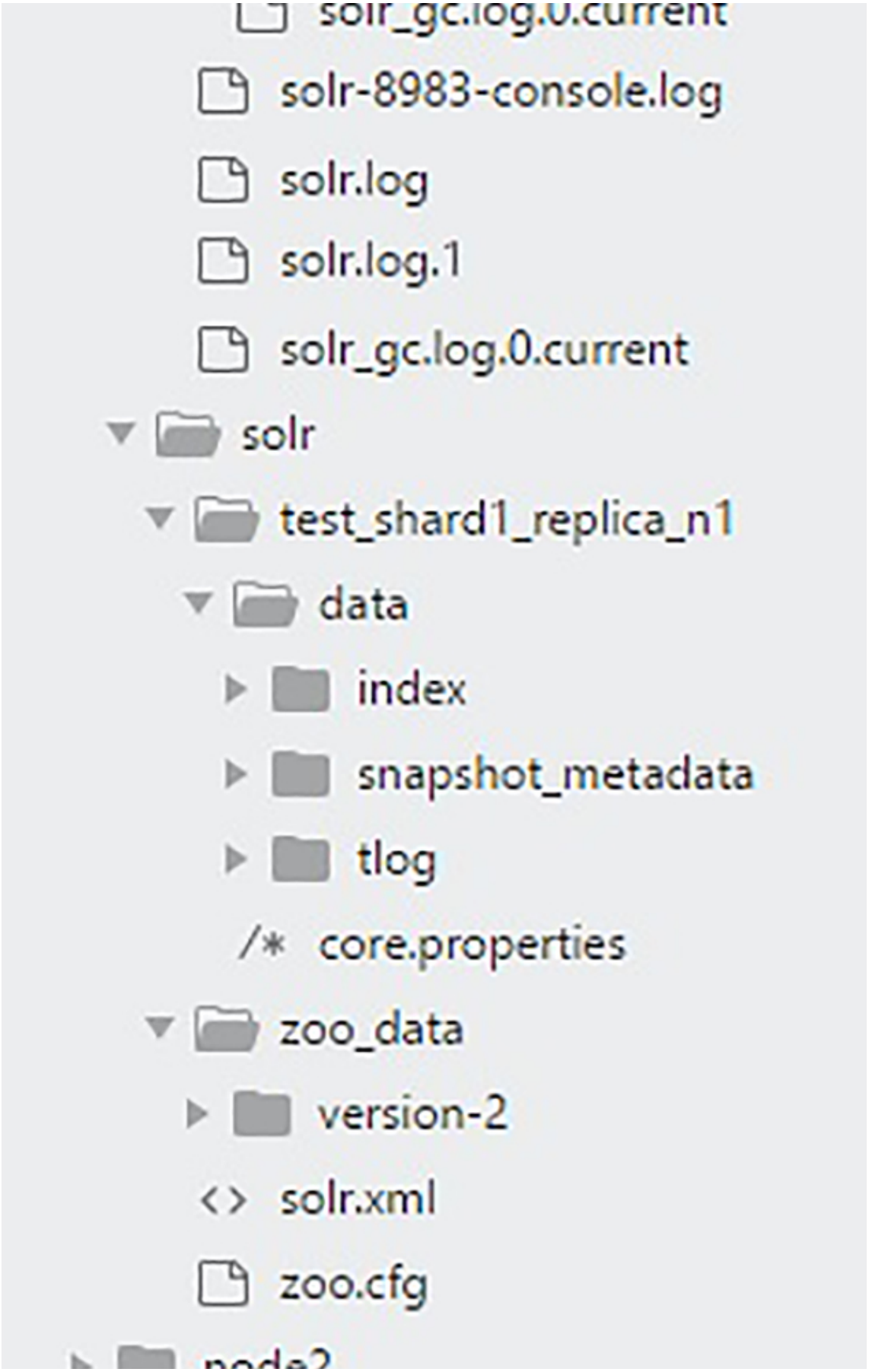
- `films`
- `files`
- `exampledocs`

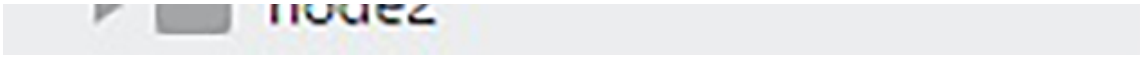
Solr provides the following config sets:

- `schemaless`
- `DIH`
- `techproducts`

Let's look at one such directory structure of an out-of-the-box example of Solr `cloud`:




solr_gc.log.0.current
solr-8983-console.log
solr.log
solr.log.1
solr_gc.log.0.current
solr
test_shard1_replica_n1
data
index
snapshot_metadata
tlog
core.properties
zoo_data
version-2
solr.xml
zoo.cfg



Here is the detailed directory structure of our out-of-the-box example of Solr `cloud`:

- The parent-level directory contains `node1` and `node2`, stating that Solr has been started in cluster mode for this config set.
- Each node contains two folders, `solr` and `logs`.
- The `solr` folder contains two folders, `test_shard1_replica_n1` and `zoo_data`. Since we have started Solr in clusters, it creates one replication folder, `test_shard1_replica_n1`. It contains the `data` folder and one file, `core.properties`, which has information about the other node.
- Apache Solr comes with embedded ZooKeeper. `zoo_data` is the ZooKeeper data directory.
- In the `solr` directory, there are two configuration files: `solr.xml` and `zoo.cfg`.
- `solr.xml` has some global configuration options that apply to all or defined cores.
- `zoo.cfg` contains the ZooKeeper-related files.

Let's look at all the configuration files that we would be using on a day-to-day basis.

core.properties

The core.properties contains some of the following properties, which we can configure as per our needs for our Solr core:

Property	Description
`name`	The name of the Solr core. Whenever you need to reference `SolrCore` with `CoreAdminHandler`, you will need this name.
`config`	Configuration filename and path. By default, this is `solrconfig.xml`.
`schema`	Schema filename of a core.
`dataDir`	Data directory where indexes are stored.
`configSet`	The config set that should be picked up to configure the core.
`properties`	The properties file path for this core.
`transient`	This decides whether the core can be unloaded or not if Solr reaches `transientCacheSize`.
`loadOnStartUp`	This decides whether or not to load the core on startup.
`coreCodeName`	A unique identifier for the node hosting the core. It can be helpful when you are replacing a machine that has had a hardware failure.
`ulogDir`	Directory for the update log.
`Shard`	The name of the shard to which the core would be assigned.
`Roles`	The name of the collection of the core in which you will index data.

zoo.cfg

This contains some of the following properties, by which we can configure for ZooKeeper:

Property	Description
`tickTime`	The `tickTime` decides actually how long `eachTick` has to be. This part of ZooKeeper determines which servers are up and running at any point in time by sending ticks.
`initLimit`	The amount of time in ticks for a forwarder to connect to the leader. It's the number of ticks that can be allowed for the initial synchronization phase to take place.
`syncLimit`	The amount of time that can be allowed for followers to keep in sync with ZooKeeper. If the followers cross this limit and yet don't sync up, they will be dropped.
`dataDir`	This is the directory in which cluster data information would be stored in ZooKeeper. It should initially be empty.
`clientPort`	This is the port at which Solr will access ZooKeeper; when this file is in place, you can start a ZooKeeper instance.
`maxClientCnxns`	The maximum number of client connections you want to handle.
`autopurge.snapRetainCount`	The number of snapshots to retain in the data directory.
`Autopurge.purgeInterval`	The purge task interval in hours. After this much time, it will start the retain task.

Note

It is not recommended to use embedded ZooKeeper in production.

solr.xml

The `solr.xml` contains Solr configuration that can apply to single or multiple cores as needed. The following table briefly summarizes the configurations available in this file:

Parent element	Property	Description
solrCloud	distribUpdateConnTimeout	Used to define the connection timeout limit for intra-cluster updates
solrCloud	distribUpdateSoTimeout	Used to define the socket time for intra-cluster updates
solrCloud	host	The hostname that Solr will use to access cores
solrCloud	hostContext	The context path
solrCloud	hostPort	The port that Solr will use to access cores
solrCloud	genericCoreNodeNames	Decides whether the node names should be based on the address of the node or not
solrCloud	zkClientTimeout	The timeout limit for connection to the ZooKeeper server
solrCloud	zkCredentialsProvider and zkACLProvider	The parameters used for ZooKeeper access control
solrCloud	leaderVoteWait	The Solr node will wait for this much time for all known replicas of that shard to be found
solrCloud	leaderConflictResolveWait	The maximum time the replica will wait to see conflicting state information to be resolved
shardHandlerFactory	socketTimeOut	The read time out for querying among intra-clusters
shardHandlerFactory	connTimeOut	The connection timeout for intra-cluster queries and administrative requests
shardHandlerFactory	urlScheme	The urlScheme to be used in distributed search
shardHandlerFactory	maxConnectionsPerHost	Maximum connections allowed per hosts
shardHandlerFactory	maxConnections	Maximum total connections allowed
shardHandlerFactory	corePoolSize	Initial core size of threadpool servicing requests
shardHandlerFactory	maximumPoolSize	Maximum size of threadpool servicing requests
shardHandlerFactory	maxThreadIdleTime	The amount of time in seconds that a thread persists in queue before getting killed

server

This directory contains an instance of the Jetty servlet and a container setup to run Solr. Given here is the `server` directory layout:

- `server/contexts` : This contains the Jetty web application deployment descriptors required by the Solr web app.
- `server/etc` : This contains Jetty configurations and has an example SSL keystore.
- `server/lib` : This has Jetty and other third-party libraries that are needed. It has folder `ext` ; any external JAR you want you can keep there.

- `server/logs` : This has the Solr log files.
- `server/solr-webapp` : This contains files used by the Solr server. As Solr is not a Java web application, don't edit the files in this directory.
- `server/solr` : This is the default `solr.home` directory where Solr will create the core directories. It must contain `solr.xml` .
- `server/resources` : This must contain configuration files such as various Log4j configurations (`log4j.properties`) required for configuring Solr loggers.
- `server/scripts/cloud-scripts` : This is the command-line utility for working with ZooKeeper when we are running Solr with SolrCloud mode. You can check out `zkcli.sh` or `zkcli.bat` for usage information.
- `server/solr/configsets` : Contains various directories; they contain configuration options essential for running Solr.
- `_default` : This has some bare minimum configurations, with the options of field guessing and managed schema turned on by default so as to start indexing data in Solr without having to design any schema upfront. Schema management can be done via the REST API as you refine your index.
- `sample_techproducts_configs` : This has example configurations that show many powerful features based on the use case of building a search solution for tech products.

Now that we have a detailed idea about Solr's directory and file structure, let us get acquainted with running and configuring Solr for our needs.

Running Solr

Let's start with running and configuring our Solr. We will see several ways of running Solr, some configurations needed for Solr to be in production mode, and more. The following topics will be covered:

- Solr startup
- Production-level Solr setup and configurations

Running basic Solr commands

Earlier, we started Solr in interactive mode, where we picked up the cloud as the default config set. Let's revisit the Solr startup commands.

If you want to start Solr with a custom port, then you can use the following command:

```
solr start -p 8984
```

Note

Based on your operating system, you have to use `bin/solr` or `bin\solr.cmd` .

Similar to `start` , there's a command `stop` . To stop Solr, simply use the following command:

```
solr stop -all
```

The `-all` parameter stops all Solr instances; if you want to stop a specific port, then pass that port number with an argument with parameter `-p` . Let's say you start Solr with any key; then you can stop that particular instance by passing the key as a parameter:

```
solr stop -k <key_name>
```

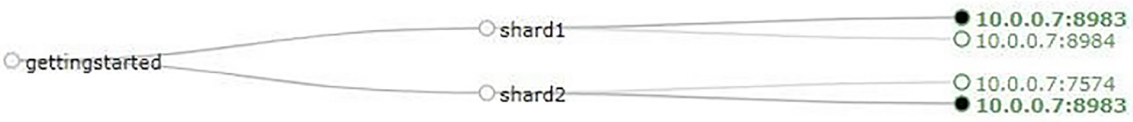
As seen earlier, you can launch examples by passing the `-e` flag:

```
solr -e <example_name>
```

Let's start Solr in cloud mode; hitting `solr -e` cloud will start the interactive process as follows:

- Specifying the number of nodes on which we would like to run Solr in the local cluster
- Port number for each Solr instance
- Creating a Solr instance home directory with node 1, node 2, and node 3
- Specifying the name for a new collection
- Specifying the shards, the new collection should split
- The replicas per shard we want to create
- The configuration for collection which we want

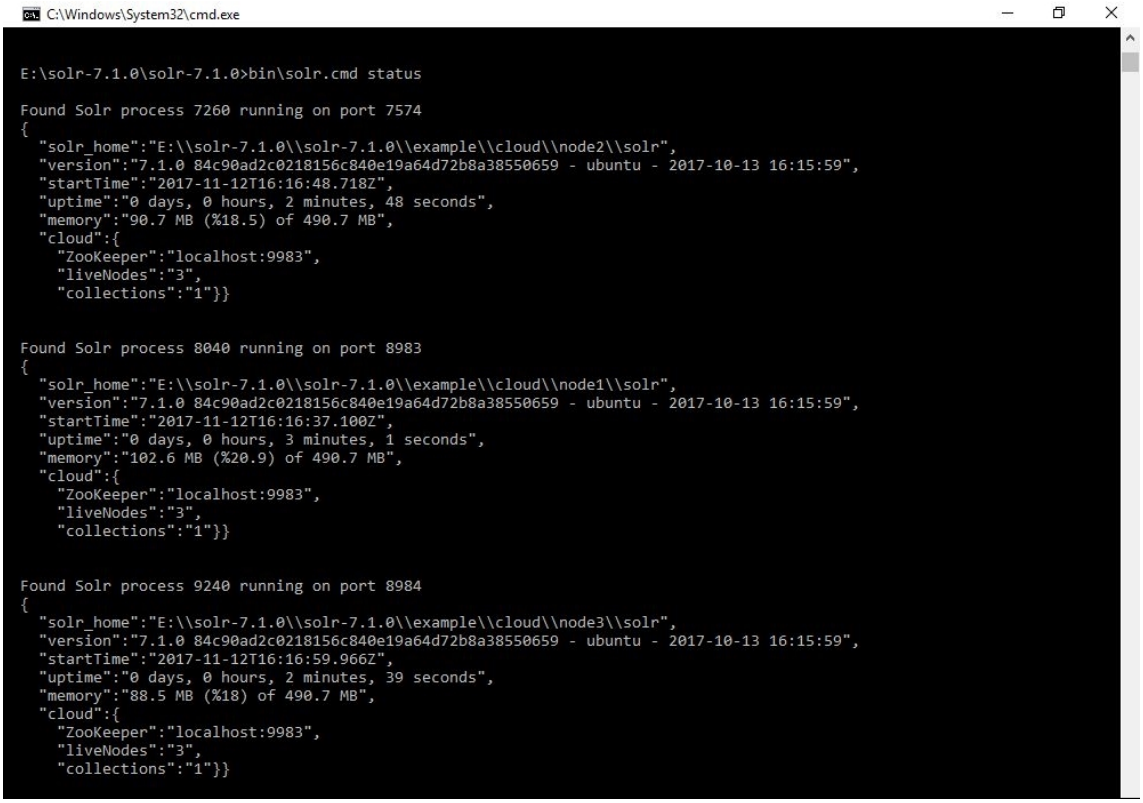
After this, the config API will be called and updated with the selected configurations. Then you can see the Solr running status on hitting `http://localhost:8983/solr/#/~cloud`. It should show something like this:



Now, to check the status of the running instances of Solr, it provides the command status. Let's check out the number of Solr instances we have up and running:

```
solr status
```

The following screenshot is the output that shows the status of the Solr service, with a few more details such as `uptime`, `solr_home`, `memory`, and so on:



Now, if you haven't started Solr with example config sets, you will find no core created. Simply put, a core is just like a database in an SQL table. To create a new core at any time, just use the following command:

```
solr create -c chintan
```

This command essentially creates a core or collection based on whether Solr is running in standalone (core) mode or SolrCloud (collection) mode. Based on the running instance of Solr, this command does its action.

Note

By default, any new core created follows the `_default` config set, which is not a recommended practice in production.

If you want to delete a collection, just invoke the following rest point:

```
 http://localhost:8983/solr/admin/collections?action=DELETE&name=<name_of_collection>
```

Now Let's add some sample documents to our collection. I have some PDF documents on microservices and Node.js, and I am indexing in Solr. Open up your terminal.

If you are on Linux, hit the following command:

```
post -c chintan /root/Desktop/apache-solr/*.pdf
```

If you are on Windows, do this:

```
java -Dauto -Dc=chintan -jar post.jar E:\\books\\solr\\fenago\\*.pdf
```

In both cases, it looks for all the PDF documents in the folder, indexes them, and adds them into the collection or core: `chintan`.

Note

If you want to use the post utility in Windows, download Cygwin and try the same command as in Linux.

Now that the documents are successfully indexed, Let's verify by asking Solr questions. Open up the terminal and query collection getting started by asking questions about the query parameter `nodejs` by hitting the following end point, which would be `http://localhost:8983/solr/gettingstarted/select?q=nodejs`.

You should get a response like this:

```
{"responseHeader":{ "zkConnected":true, "status":0, "QTime":22, "params":{
"q":"nodejs"}}, "response":{"numFound":2,"start":0,"maxScore":2.283722,"docs":[ {},{}]
```

Production Solr setup

Now that we are done playing around in Solr, Let's take a step further and look into installing Solr as a service. This is specifically useful in servers and Linux systems. Follow these steps:

1. The Solr bundle you downloaded earlier contains the `solr` directory, which includes a shell script `bin/install_solr_service.sh`. This will help to install Solr as a service on Linux. As some prerequisites, it will need the path where you want to install Solr and the user who should be owning Solr, that is, privileges.
2. While installing Solr, make sure that the `indexed` directory and `logs` directory stay out of the `solr` installation folder; this will be very helpful when we upgrade Solr.
3. The Solr service script, by default, extracts the archive in `/opt`. If you want to change that path, just add the `-i` option while running the Solr script. You can install the Solr service by simply hitting:

NOTE

Solr has been already installed "/opt/solr", their is no need to run following setup again.

```
sudo bash ./install_solr_service.sh solr-8.6.0.tgz
```

It will install in the `opt` folder and also create a symbolic link:

```
/opt/solr --> /opt/solr-8.6.0
```

This helps in easy and smooth upgrades.

4. The data directory should be different; the script takes care of it. By default, it adds a directory at `/var/solr`, but if you want to change that, you can pass a location using `-d`.
5. Running Solr as a root user is not recommended. This script creates a Solr user by default, so `/var/solr` and `/opt/solr` can be fully owned by the Solr admin. No need to give them root privileges!

Now that the script is installed, you can start/stop Solr at any time with the following commands:

```
sudo service solr start
sudo service solr stop
```

So after getting up and running with basic utilities in Solr, covering the various available options in Solr, and getting to install and run a Solr service in Linux, let us now play with data. We will load some sample data and understand

the various ways to load and query data.

Loading sample data

Now that we've got acquainted with Solr and the various commands involved in Solr's day-to-day usage, Let's populate it with data so that we can query it as needed. Solr comes with sample data in examples. We will use the same `$solr_home/example/films` for our queries.

Fire up the terminal and create a collection `films` with `10` shards:

```
solr\bin create -c films -shards 10
```

Now, in `$solr_home/example/films` there is file called `file.json`. Let's import it into our collection, `films`. Based on your OS, hit the appropriate command for the post script or `post.jar`.

Uh Oh!! It throws an error, as follows:

What must have gone wrong? By checking the logs while creating the collection, you must have seen a warning.

Note

`Warning` Using `_default` config set data driven schema functionality is enabled by default, which is not recommended for production use.

To turn it off, use the following command:

```
curl http://localhost:8983/solr/test11/config -d '{"set-user-property":
{"update.autoCreateFields":"false"}}'
```

While creating the collection, we went on with the `_default` config set. The `_default` config set does two things: we make use of the managed schema meaning, the schema can be modified only through Solr's schema API. We don't specify the field mapping and let the config set to do the guessing. It may seem advantageous at one point because we don't have to restrict Solr to know any pre-fields and we can adopt the concept of schemaless. Solr will create fields on demand as it encounters documents. Now, this very advantage had become a problem. If you open up `films.json` and check the name of the first field, you will see it as `.45`. Solr, guessing it as Float type, keeps the datatype of the filmname as Float; the moment it encounters text, it spits out the error. We end up with huge trouble as we can't change the mapping fields once the index contains data.

For the same reason, it is not recommended to go with the `_default` schema config set. So, Let's solve this issue by leveraging the schema API to modify our schema definition.

Delete the films collection by hitting the rest point:

```
http://localhost:8983/solr/admin/collections?action=DELETE&name=films
```

Create the collection again using the following command:

```
solr -c films -shards 10 -n schemaless
```

Note

We will be passing `-n` so that it will pick up the schemaless configuration instead of the previous default configuration.

Now make a post call with the following details, which will basically tell Solr to expect a field's name as a text rather than letting it auto-guess as Float:

End point `http://localhost:8983/solr/films/schema`

Header `{"Content-Type":application/json}`

Body `{"add-field": {"name":"name", "type":"text_general", "multiValued":false, "stored":true}}`

You should get a successful response.

Now try hitting the import command again: `java -Dauto -Dc=films -jar post.jar E:\solr-8.6.0\solr-8.6.0\example\films\films.json`

You should be able to do so successfully. Go to the browser and open `http://localhost:8983/solr/films/select?q=*:*`. You should be able to see 1,100 records. You can similarly import CSV and XML files.

While importing a CSV file, you need to specify that if a field has multiple values, it should split; and you need to specify what the separator would be. If you check out `films.csv`, then you will notice that genres and `directed_by` are such fields. In this case, our import command would be:

```
java -jar -Dc=films -Dparams=f.genre.split=true & f.directed_by.split=true &
f.genre.separator=| & f.directed_by.separator=| -Dauto example\exampledocs\post.jar
example\films\*.csv
```

This is how we can load unstructured data in Solr. Let's now look at how to load structured data in Solr.

Loading data from MySQL

Solr's contrib provides the `datahandlerimport` module, and one of the examples in Solr is also focused on `DataImportHandler`, also known as DIH. Let's run the DIH example given by default. Hit `solr -e dih` to start Solr with the example of DIH. It will pick the configuration set in `$solr_home/example/example-DIH`. Create sample data in MySQL as follows:

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input sta

mysql> use parth;
Database changed
mysql> desc archives;
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| category_id   | int(11)     | NO   | PRI | NULL    | auto_increment |
| category_name | varchar(150)| YES  |     | NULL    |                |
| remarks       | varchar(500)| YES  |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> select * from archives;
+-------------+---------------+-------------+
| category_id | category_name | remarks     |
+-------------+---------------+-------------+
|           1 | parth         | test test   |
|           2 | ghiya         | test1 test1 |
|           3 | random        | test1 test1 |
|           4 | perplex       | test1 test1 |
|           5 | mystified     | test1 test1 |
+-------------+---------------+-------------+
5 rows in set (0.00 sec)
```

Now it's time to integrate it into Solr. Follow these steps to find out the necessary configurations:

1. As stated earlier, any Solr installation would require the `solrconfig.xml` file, which has the necessary config information. This file usually resides in `$solr_home/<config_set_selected>/conf`. As we have selected `-e dih` and we are specifically looking for the database utility, we will find our `solrconfig.xml` at `$solr_home/example/example-DIH/solr/db/conf/solrconfig.xml`. Add the following lines of code in this file.

Since we are going to use MySql, we have downloaded the MySQL connector JAR, place it in the `dist` folder `$solr_home/dist`, and let Solr know where to find the connector JAR by adding the following lines of code. Make sure that the `dataimporthandler` module is available in `dist`:

```
<lib dir="${solr.install.dir:../../../..}/dist/" regex="solr-
dataimporthandler-.*\.jar" />
<lib dir="${solr.install.dir:../../../..}/contrib/extraction/lib" regex=".*\.jar" />
<lib dir="${solr.install.dir:../../../..}/dist" regex="mysql-connector-java-\d.*\.jar"
/>
```

2. Next, we need to tell Solr where our database configuration file is. Add the following lines of code and create a file `db-data-config.xml` parallel to `solrconfig.xml`:

```
<requestHandler name="/dataimport" class="solr.DataImportHandler">
    <lst name="defaults">
```

```
      <str name="config">db-data-config.xml</str>
    </lst>
  </requestHandler>
```

3. `db-data-config.xml` is the file where we will define our database and Solr mapping. Create one file and define that file with the following schema:

```
<dataConfig>
    <dataSource driver="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/chintan" user="root" password="root" />
         <document>
         <entity name="archives" query="select * from archives"
 deltaImportQuery="SELECT * from archives WHERE category_id='${dih.delta.id}'">
 <field column="category_id" name="category_id" />
 <field column="category_name" name="category_name" />
 <field column="remarks" name="remarks" />
 </entity>
 </document>
 </dataConfig>
```

4. The last part is to let Solr know about the new entity we have added. Go to `${solr_home}/example/example-DIH/solr/db/conf/managed-schema` and make the following changes. Change the primary key from `id` to `category_id`:

```
<field name="category_id" type="string" indexed="true" stored="true" required="true"
multiValued="false" />
 …
 <uniqueKey>category_id</uniqueKey>
```

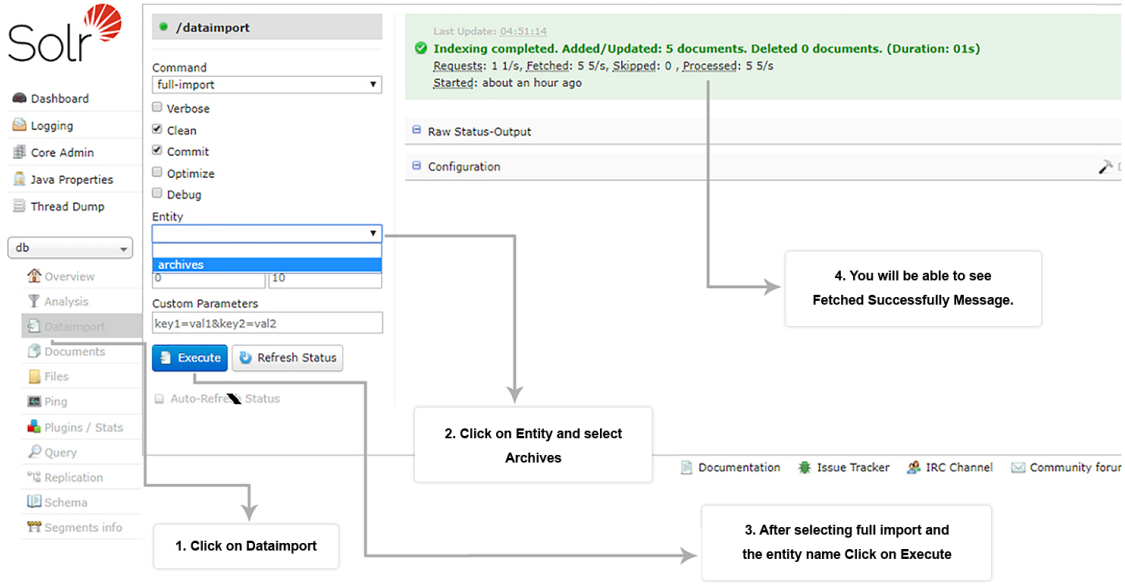
Note

This is just a method to show how to import data from MySQL, and hence we are changing the primary key. In the real world, doing this is strongly not recommended. 5. Now Let's add the new fields you introduced in `db-data-config.xml`:

```
<field name="category_name" type="string" indexed="true" stored="true"/>
<field name="remarks" type="string" indexed="true" stored="true"/>
```

That's done. Now restart the server by hitting `solr/bin stop -all && solr/bin start -e dih`.

Go to the Solr admin panel. On the core selector, select the database and follow the steps shown in this screenshot:



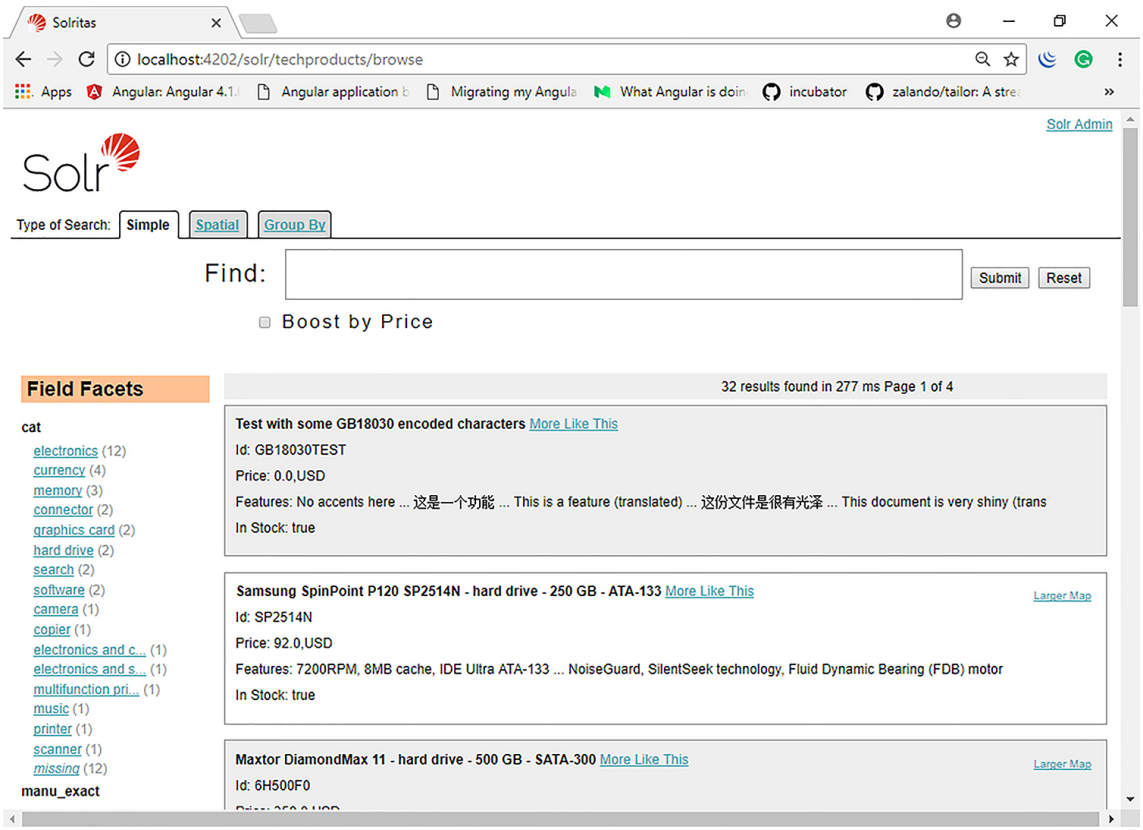
You can hit on the browser to see MySQL data in Solr.

Let's look at ways to browse data in Solr through the browse interface.

Understanding the browse interface

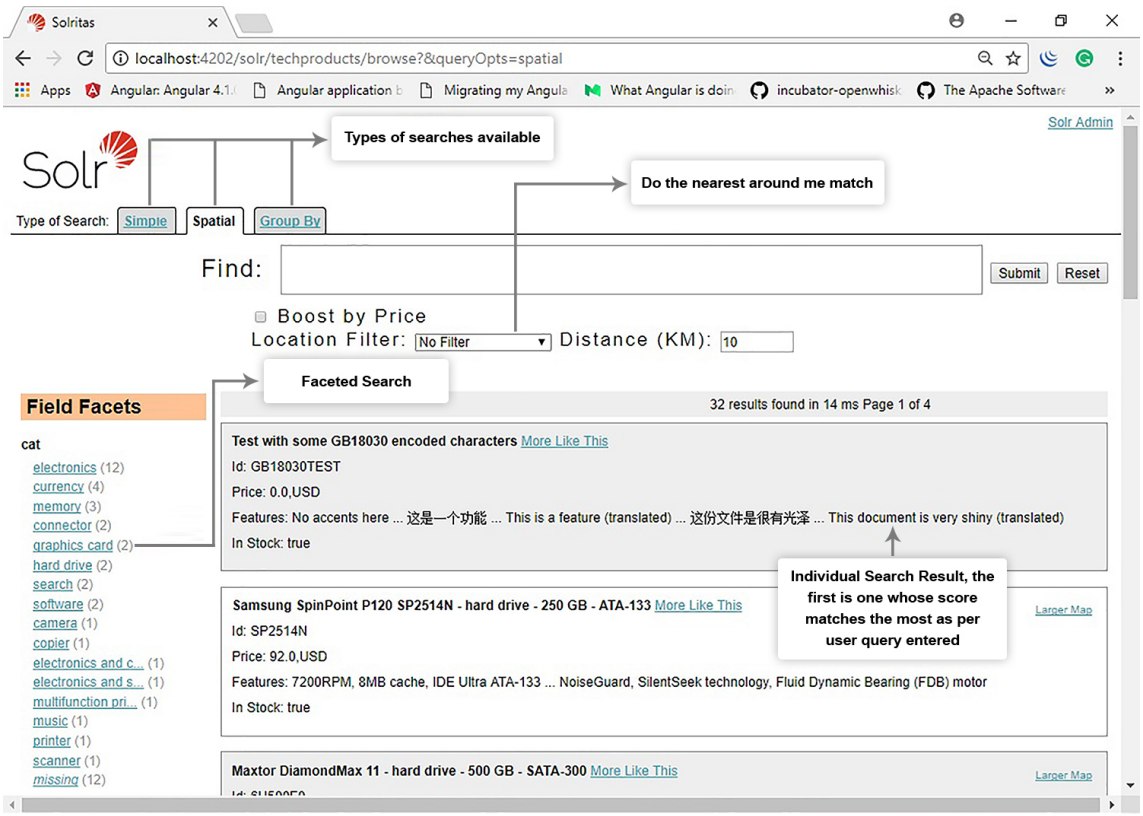
Now that our Solr is loaded up with data, we will look at multiple queries and the browse interface, through which we can query without actually knowing the end points. The data provided in `techproducts` includes a wide variety of fields along with geospatial indexes. So Let's use that for a change. Open up the terminal and hit `solr -e techproducts -p 4202`. As we have loaded a sample `techproducts` config set, it will import a bunch of files into the collection while starting up the server.

Once the server is up and running, hit `http://localhost:4202/solr/techproducts/browse` to check out the browse interface provided by Solr, as shown in the following screenshot:



It is just another Google search for electronics now. Go ahead and type the query string parameter. It will autocomplete and display the results as needed.

The following screenshot explains the browse interface in a lot of detail:



This feature makes use of the solariatis velocity contrib, which we saw earlier. This is useful for generating textual (it may or may not be HTML) from the Solr request. Using the Solr admin interface

---

Solr provides a web interface for feasibility of Solr administrators and programmers to perform the following operations easily:

- View Solr configuration details
- Run different queries against indexes
- Analyze document fields to fine-tune the Solr config set
- Provide a schema browser for easy querying
- Show the java properties of each core

And much more... Under the hood, Solr reuses the same HTTP APIs that are available to all clients. Let us look at the Solr admin panel in detail. Go and start up Solr with the config set we created earlier. Accessing `http://localhost:7574/solr` or `http://localhost:8983/solr` will open up the Solr admin dashboard. The whole screen is divided into two parts:

- The left part showing the navigational menu
- The right part showing the interface selected menu

Let's take a deep dive into each of the sections.

Dashboard

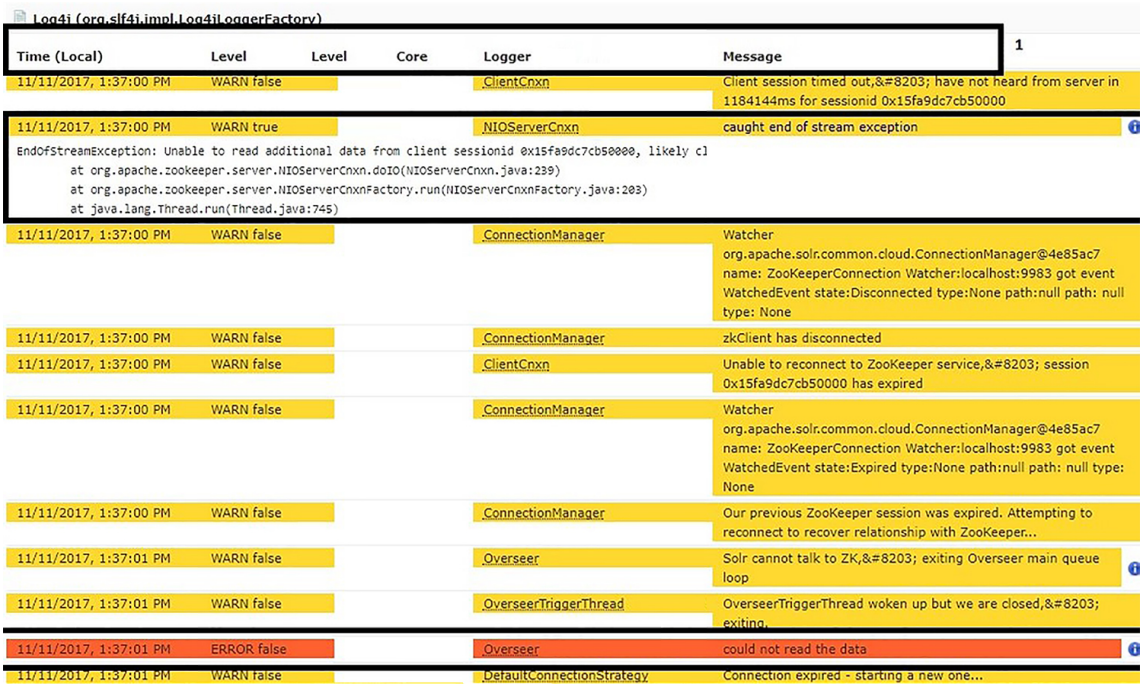
This is the default page that opens up whenever we open Solr. The dashboard contains various pieces of information:

- **Instance details**: Tells us when the instance was started.
- **Versions information**: This gives us detailed information about the Solr and Lucene specs and implementation versions.

- **JVM information**: This details out information about JVM. It covers the JVM runtime environment found, processors, and arguments supplied to Solr. All the configurational arguments passed in various Solr config files can be seen combined here.
- **System**: This shows a pictorial view of the system status occupied right now and how much is available. The first and last gray bars show the physical and JVM memory available. The first is a measure of the amount of memory available in the hosting machine. The second shows the amount assigned to the startup of Solr in units of `-Xms` and `-Xmx` options.
- **JVM memory**: This shows info about the JVM memory and the memory and percentage that Solr occupies at any point in time.

Logging

Logs are a way to know what's going in the system at any point in time. When you click on logging, a screen similar to the following screenshot comes up. This is the interface where real-time logs of Solr are shown. It even supports multi-core logs. You can see various log levels, time, message, the core from which it comes, and so on:



Here are a few highlights of the logging details of our out-of-the-box Solr example:

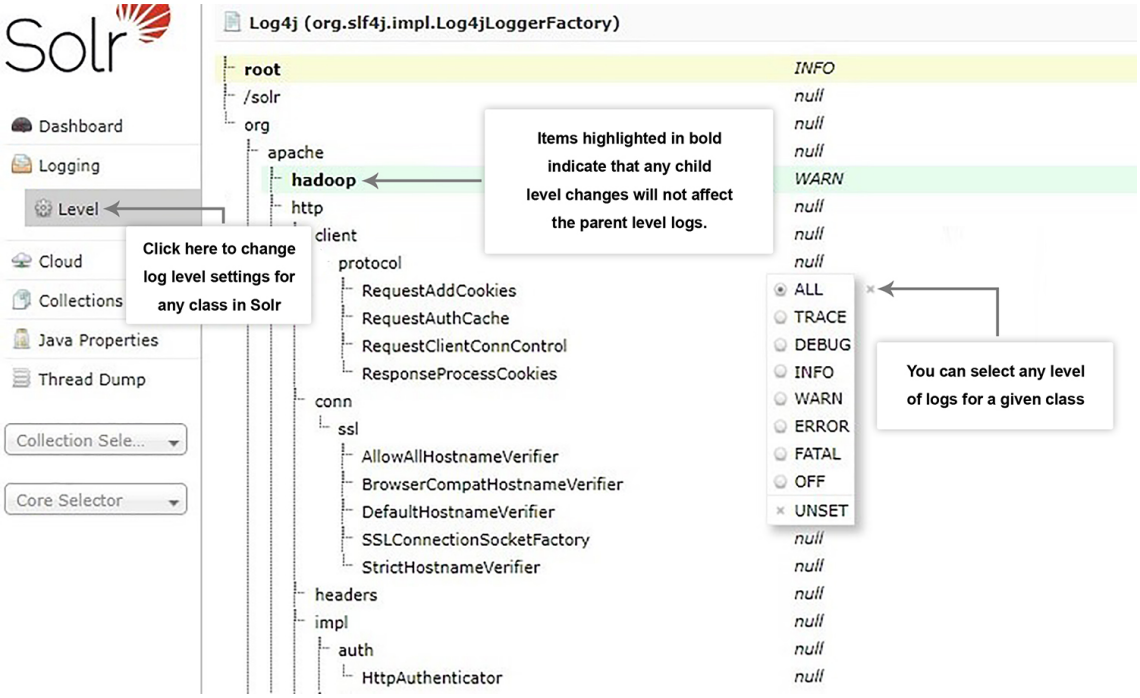
- It shows logs based on time, level, core, logger class, and message.
- Some of the messages have an informative icon at the end. Clicking on it prints the stack trace.
- There are different log levels. The red ones are error level logs, and the yellow ones are warning level logs.

Solr provides a way to change the log level at any running system instance. You can adjust the level of logs for any class. The various options for the log level you can select are all, trace, debug, info, warn, error, fatal, off, and unset.

You can change the log levels of any class at any point of the running instance.

When you open up the level in Solr, you will see all of the hierarchy of class paths and class names. Any class that has logging capabilities would be marked in yellow. Click on the highlighted row and you will be able to change the log level of that class.

One interesting parameter you will see is unset. Any category that is unset will have log levels of its parents. This simple feature allows us to change many categories at once by just changing the log level of the parent:



Note

This is a runtime setting and is not persisted, so if you open Solr again, it will be lost.

This isn't the only way we can change the log levels. Log levels can also be changed in the following ways:

- Using the log level API as mentioned here:

```
# Set the root logger to level WARN curl -s
http://localhost:8983/solr/admin/info/logging --data-binary "set=root:WARN"
```

- Log levels can also be selected at startup. Again, this can be done in two ways:
    - Search for the string `SOLR_LOGS_LEVEL` in `bin/solr.in.cmd` or `bin\solr.in.sh`. Change it as needed.
    - The second alternative is to pass at startup with the `-v` or `-q` options. For example:

```
solr start -f -v
solr start -f -q
```

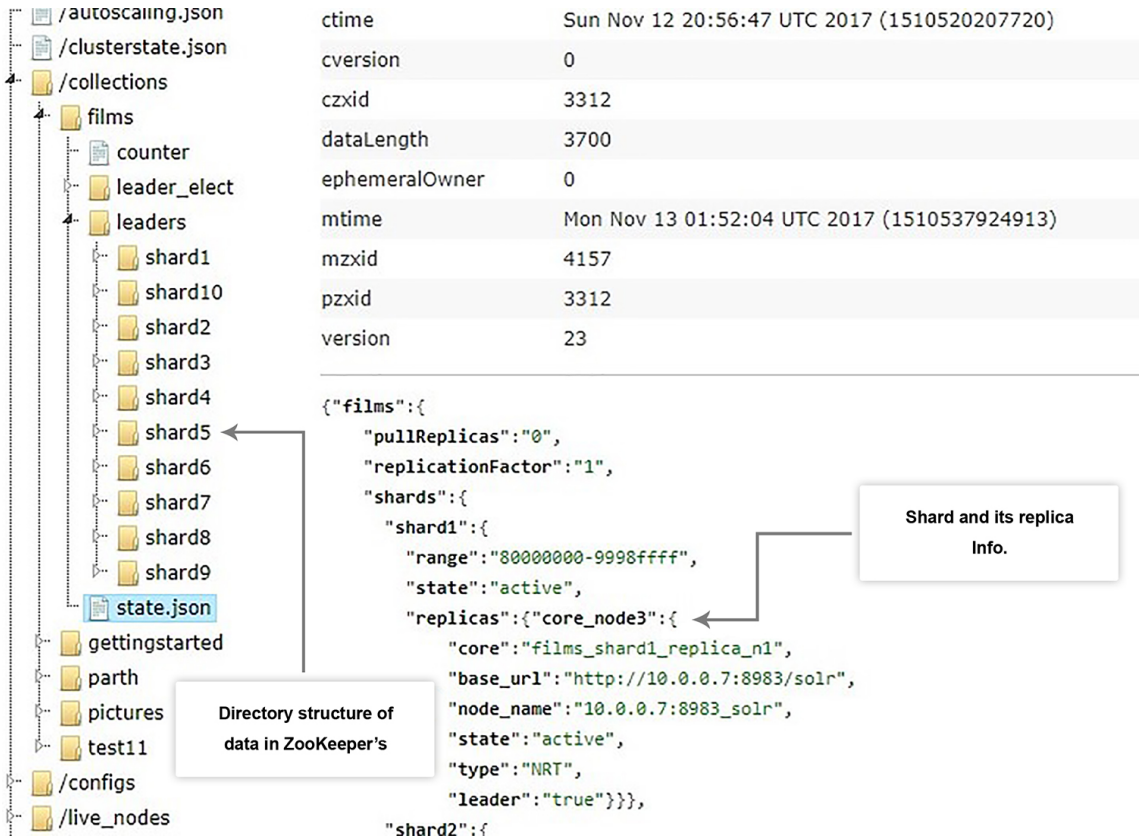
- A more permanent solution can be to change `log4j.properties` directly, which can be found at `$solr_home/server/resources`.

Cloud screens

When we run Solr in cloud mode, a cloud option will appear between logging and collections. This screen gives us the details of each collection and node in the cluster. It gives information about the level data stored in ZooKeeper. An important point to note is that this option is not visible when a single node is running or master-slave replication instances of Solr are running. It provides three different views: tree, graph, and graph (radial).

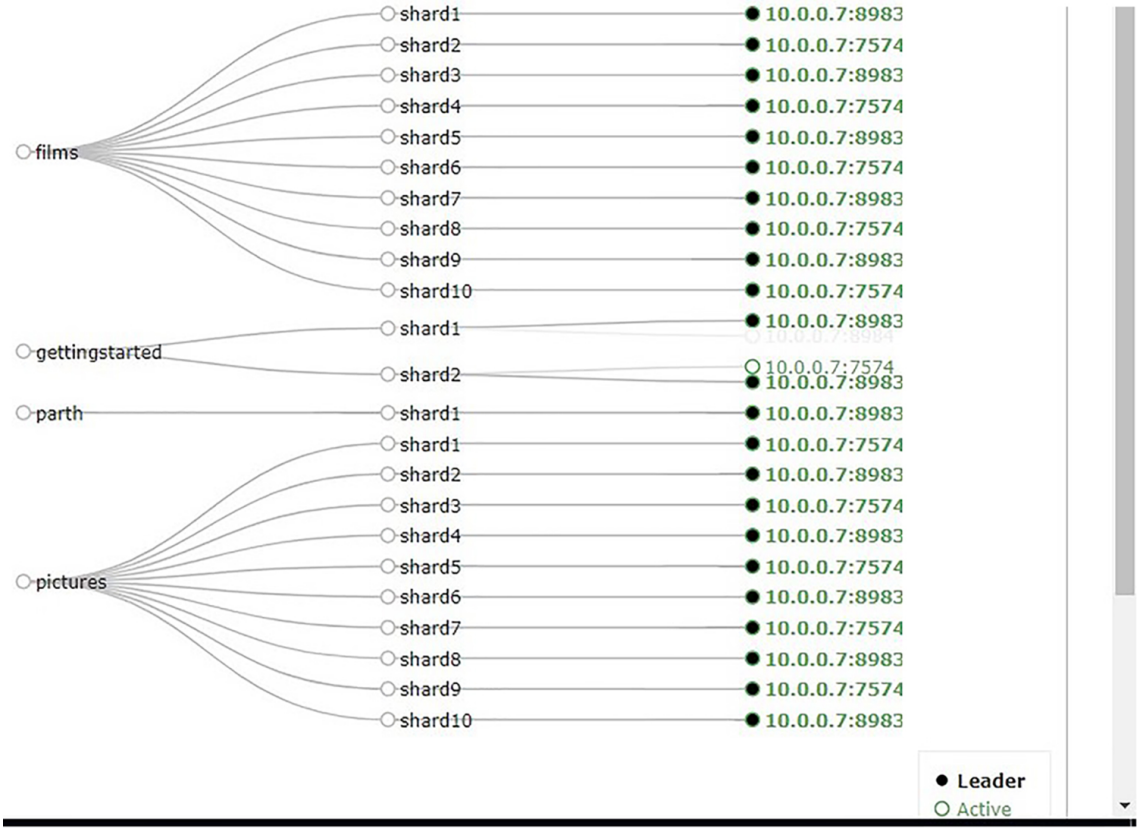
Tree view

This shows the directory structure of data residing in ZooKeeper and according to ZooKeeper configurations. It has cluster-wide information such as the number of live nodes, overseer, and overseer election status. It also has collection-specific information, such as `state.json` (which has a definition of the collection), the shard leaders, and the configuration files in use.



Graph view

This shows a graph of each collection, the number of shards that constitute that collection, and the address of each of the replicas of that shard. At the bottom of the screen, it shows labels for the leader shard, active shards, recovering shards, failed shards, inactive shards, and gone shards:



As you can see, there are multiple collections: `films`, `gettingstarted`, `chintan`, `pictures`, and so on. Most of the shards are active but notice a difference in the `gettingstarted` collection. Its shard 2 node has one blacked-out circle and another empty circle. This means that it is just an active node and not the leader. Also, if you check out shard 1 of `gettingstarted`, you will see a faint color; this means that this shard is no longer active.

Now Let's look at some of the core admin stuff that we will be using in day-to-day life. The next section of collections is just a visual interface for the available collections API. Behind the scenes, this interface calls the very same APIs exposed over the rest. As a matter of fact, the Solr admin UI is made in AngularJS.

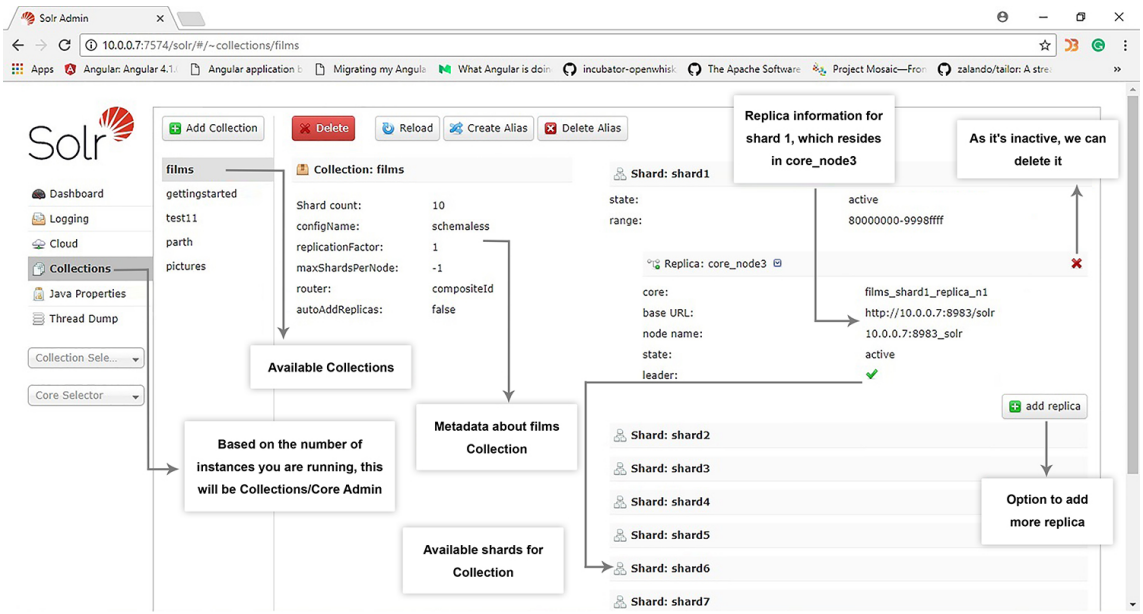
Collections or core admin

This screen provides some of the basic functionalities for managing collections in Solr, which runs the collections API under the hood.

Note

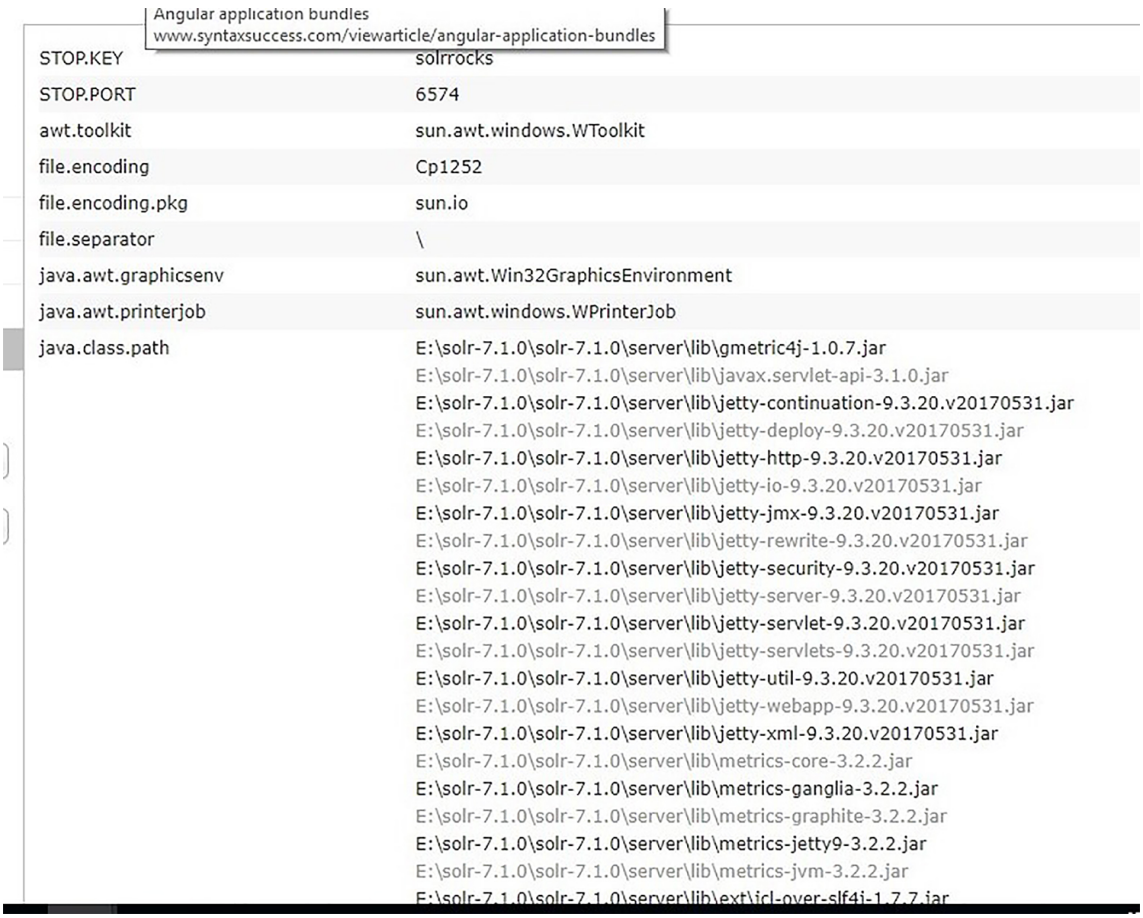
Based on the number of instances you are running, you will see the option as collections or core admin. If you are running a single instance, you will see the option as core admin, which runs `CoreAdminApi` under the hood.

Clicking on `Collections` for the first time opens the list of collections you have. Clicking on any of the collections will give you metadata about it, which has various options such as shard count, config set name, and so on. It enables various options, as seen in this diagram:



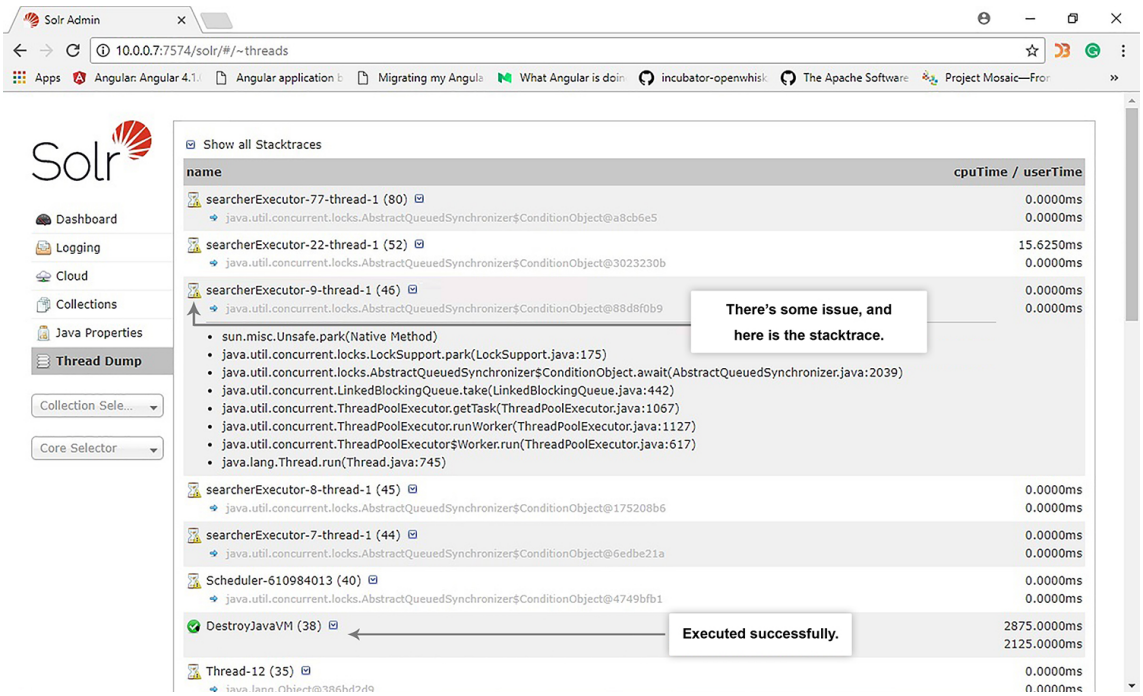
Java properties

With this screen, you can see all configured properties of JVM, which is running Solr. It includes the class path, encoding type, external libraries, and so on.



Thread dump

This is the screen that lets you view and analyze the active threads on the server in Solr. Each thread is mentioned with a number and the stacktrace access if applicable. Each icon preceding the thread name indicates the state of the thread. A thread can be in any one of `new`, `runnable`, `blocked`, `waiting`, `timed_waiting` or `terminated` states.



Collection-specific tools

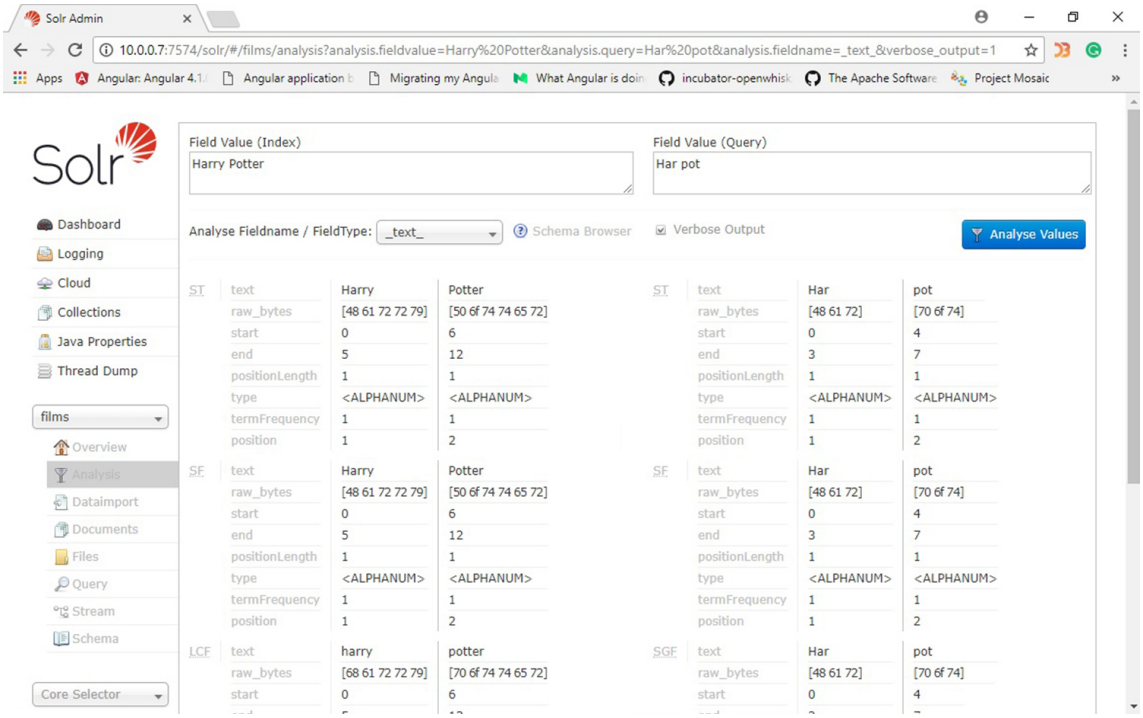
Clicking on any one of the collections in the collection dropdown opens up the various things that we can do on any collection. It has the following suboptions.

Overview

This just contains basic metadata about the collection: the number of shards, the replica per shard, the range of each shard, the config set of the instance, and whether auto-add replicas is enabled or not.

Analysis

This is the screen that helps us analyze data in the collections. We can inspect the field, field type, and configurations in our schema; furthermore, we will be able to know how the content would be applied during indexing or query processing. Let's analyze one such query that gives us a detailed output, as follows:



This screen is useful to study the analyzers applied on a collection. Analysis can be done in various phases where transformations such as uppercase, lowercase, singular/plural, synonyms, tenses, and so on will be taken into consideration.

DataImport

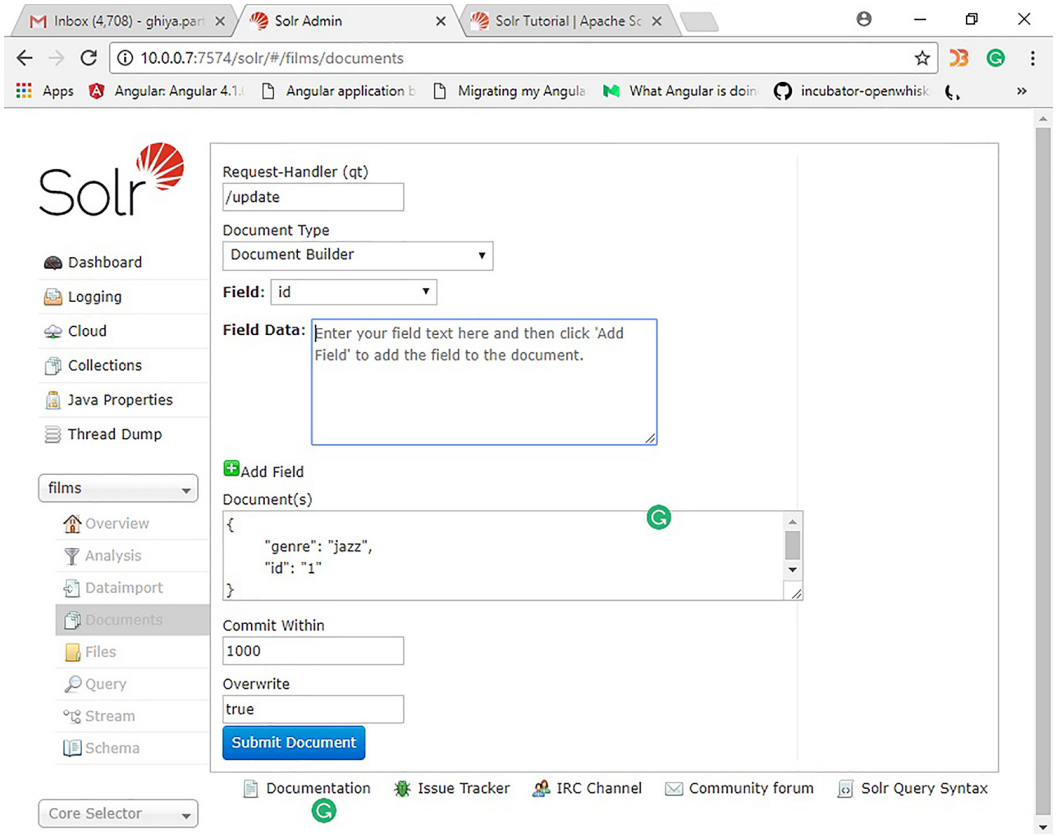
We saw this screen when we imported from MySQL. This screen allows us to monitor the status of all the import commands and the entities that we have defined in `managed_schema`.

Documents

This screen allows us to directly run various Solr indexing commands right from the browser. You can do the following set of tasks:

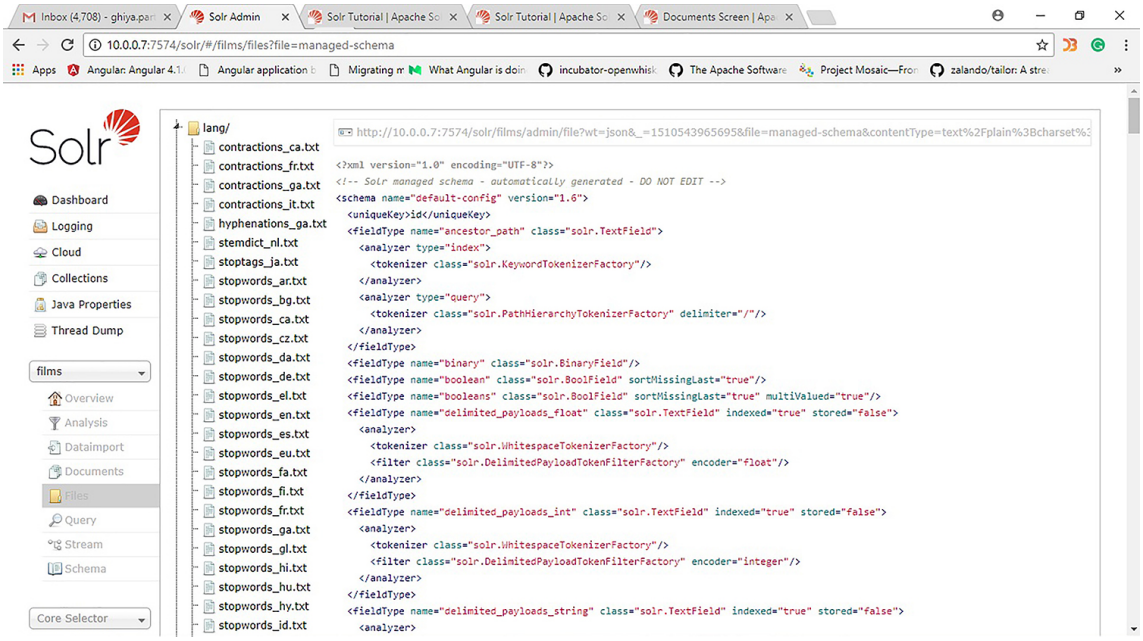
1. Copy documents in any format, select the document type, and then index
2. Upload documents

3. Construct documents by the process of selecting field type and field values



Files

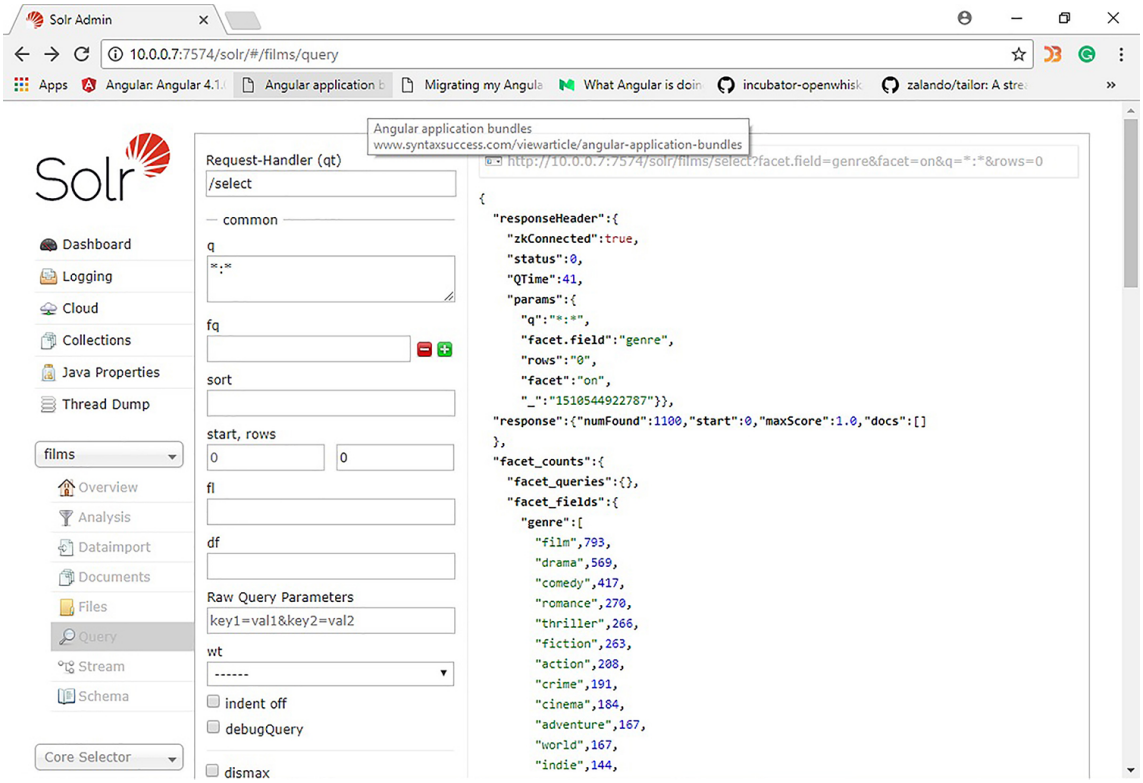
The **`Files`** screen lets us browse and view various configuration and language-related files. These files cannot be edited with this screen. If you want to edit them, you have to visit the **`Schema Browser`** screen.



Query

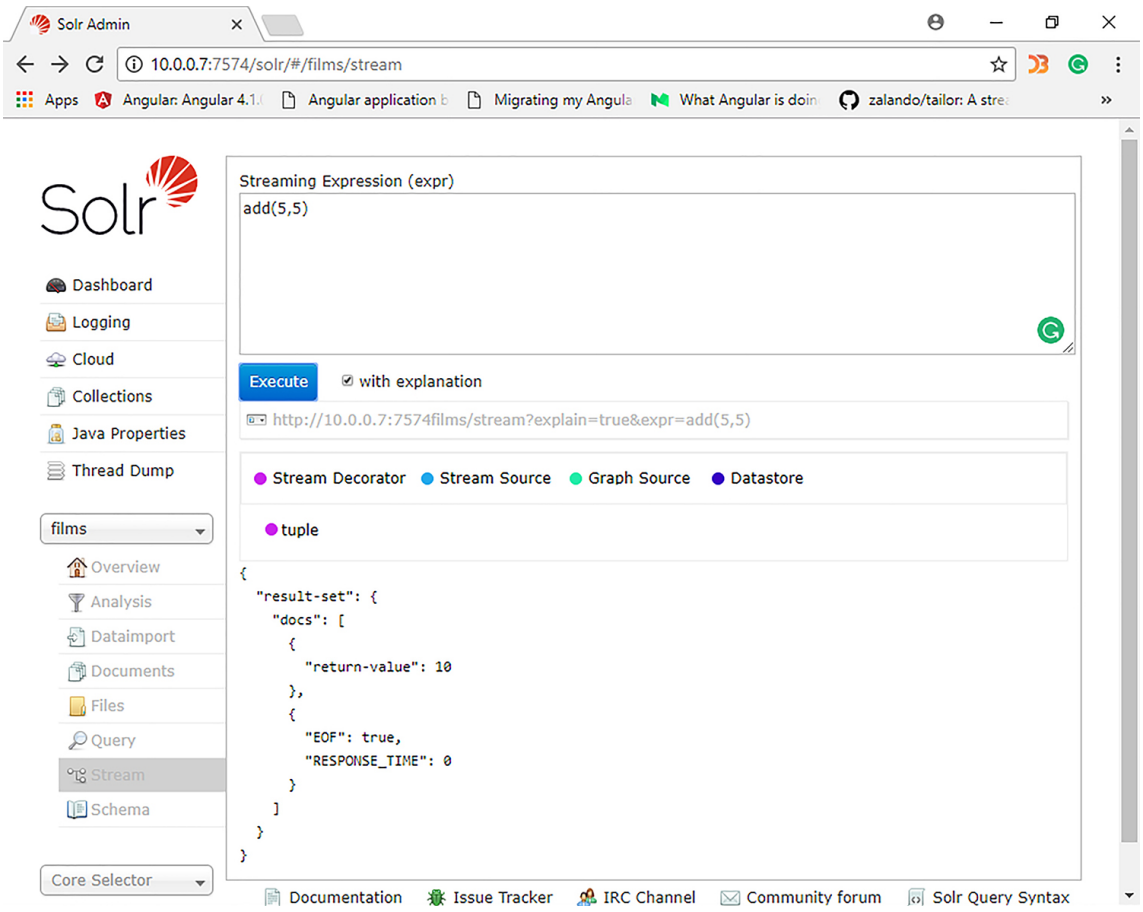
You use this screen to query your existing collections. Various kinds of queries are available in Solr, right from normal queries to geospatial queries and faceted search.

Let's do a simple faceted query on genres in our `films` collection. Go to the **`Query`** screen; at the bottom, click on **`facet`**, and in `facet.field`, enter `genre`. Click on **`execute query`**:



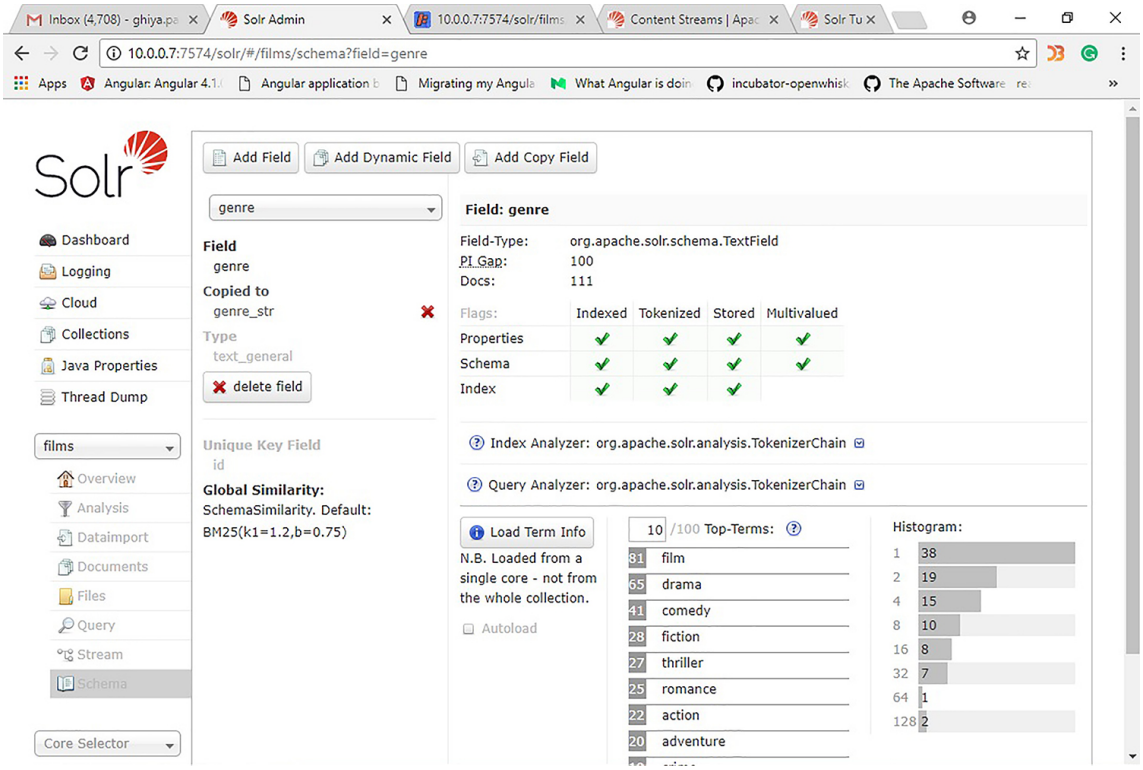
Stream

The streaming screen helps you understand the explanation of a query. It is very similar to the **`Query`** screen; it just adds an explanation part.



Schema

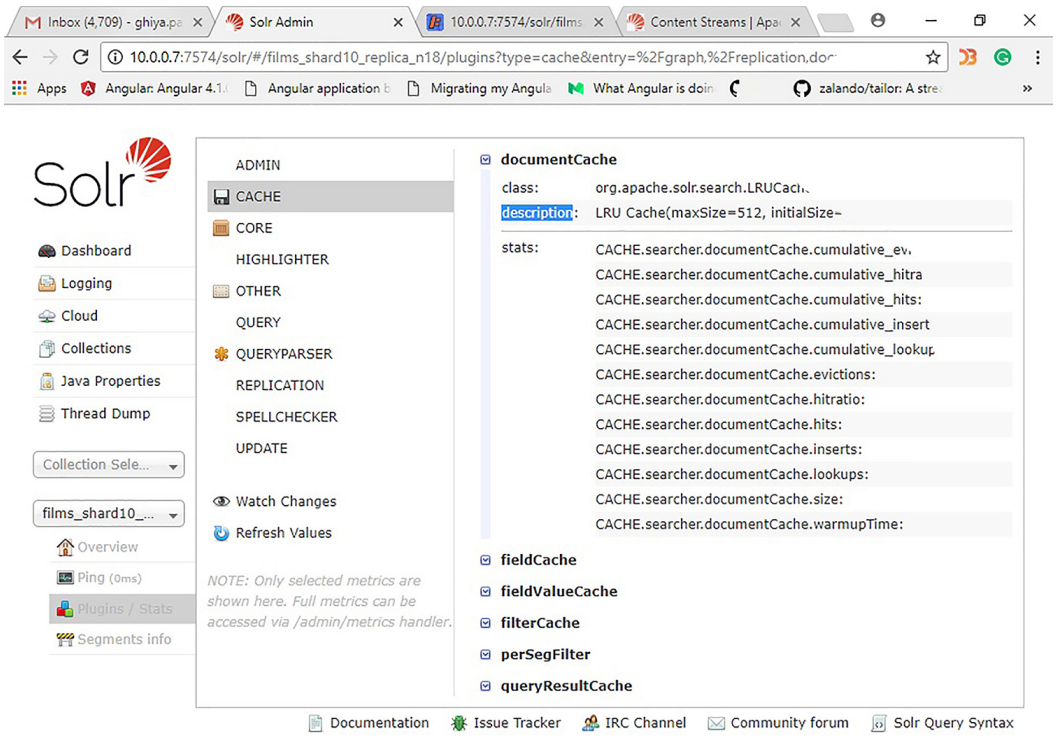
This screen lets us view the schema in a browser window. It provides in-depth information about each of the fields and its field type. It has options to add dynamic fields and copy field mapping from one field to another.



Core-specific tools

These are a group of UI utilities that give information about the core level. On selecting a **`Core`** in the dropdown, you will see the following screens:

- **`Overview`** : This will display some of the basic metadata about the running Solr core. You can add a file `admin-extra.html` , which consists of additional information if you would like to display in the **`Admin Extra block.`**
- **`Ping`** : This lets you send a ping to the selected core to determine whether the core is active or not.
- **`Plugins / Stats`** : Shows all the plugins installed and statistics. The performance factor of the Solr cache can be checked here.



- `Segments info` : This lets you see visualizations of different segments by Lucene for the core selected. It shows information about the size of each segment, with units in both bytes and number of documents. You can also check out the number of deleted documents.

Summary

In this lab, we got started with Solr. We saw the various options available to start Solr and saw the directory and folder structure of Solr in detail. We learned how to run Solr as service, Solr and ZooKeeper configurations, how to load some sample data from documents and databases, the browse interface, and the Admin UI in detail.

Let's move on to designing a schema in the next lab. We will learn how to design it using documents and fields. We will go through various field types and see the schema API. We will also look at the schemaless mode.