Community Experience Distilled

Web Crawling and Data Mining with Apache Nutch

Perform web crawling and apply data mining in your application

ATTUNE INFOCOM
Tune into Enterprise Open Source

Dr. Zakir Laliwala
Abdulbasit Shaikh

[PACKT] PUBLISHING open source* community experience distilled

Web Crawling and Data Mining with Apache Nutch

Perform web crawling and apply data mining in your application

Dr. Zakir Laliwala

Abdulbasit Shaikh

[PACKT] PUBLISHING

open source
community experience distilled

BIRMINGHAM - MUMBAI

Web Crawling and Data Mining with Apache Nutch

First published: December 2013

Production Reference: 1171213

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-685-0

www.packtpub.com

Cover Image by Jarek Blaminsky (milak6@wp.pl)

Credits

Authors
Dr. Zakir Laliwala
Abdulbasit Shaikh

Reviewers
Mark Kerzner
Shriram Sridharan

Acquisition Editors
Neha Nagwekar
Vinay V. Argekar

Commissioning Editor
Deepika Singh

Technical Editors
Vrinda Nitesh Bhosale
Anita Nayak
Harshad Vairat

Copy Editors
Roshni Banerjee
Mradula Hegde
Sayanee Mukherjee
Deepa Nambiar

Project Coordinator
Ankita Goenka

Proofreaders
Ameesha Green
Bernadette Watkins

Indexer
Mariammal Chettiyar

Graphics
Disha Haria

Production Coordinator
Conidon Miranda

Cover Work
Conidon Miranda

About the Authors

Dr. Zakir Laliwala is an entrepreneur, an open source specialist, and a hands-on CTO at Attune Infocom. Attune Infocom provides enterprise open source solutions and services for SOA, BPM, ESB, Portal, cloud computing, and ECM. At Attune Infocom, he is responsible for product development and the delivery of solutions and services. He explores new enterprise open source technologies and defines architecture, roadmaps, and best practices. He has provided consultations and training to corporations around the world on various open source technologies such as Mule ESB, Activiti BPM, JBoss jBPM and Drools, Liferay Portal, Alfresco ECM, JBoss SOA, and cloud computing.

He received a Ph.D. in Information and Communication Technology from Dhirubhai Ambani Institute of Information and Communication Technology. He was an adjunct faculty at Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT), and he taught Master's degree students at CEPT.

He has published many research papers on web services, SOA, grid computing, and the semantic web in IEEE, and has participated in ACM International Conferences. He serves as a reviewer at various international conferences and journals. He has also published book chapters and written books on open source technologies.
He was a co-author of the books *Mule ESB Cookbook* and *Activiti5 Business Process Management Beginner's Guide, Packt Publishing*.

Abdulbasit Shaikh has more than two years of experience in the IT industry. He completed his Masters' degree from the Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT). He has a lot of experience in open source technologies. He has worked on a number of open source technologies, such as Apache Hadoop, Apache Solr, Apache ZooKeeper, Apache Mahout, Apache Nutch, and Liferay. He has provided training on Apache Nutch, Apache Hadoop, Apache Mahout, and AWS architect. He is currently working on the OpenStack technology. He has also delivered projects and training on open source technologies. He has a very good knowledge of cloud computing, such as AWS and Microsoft Azure, as he has successfully delivered many projects in cloud computing.

He is a very enthusiastic and active person when he is working on a project or delivering a project. Currently, he is working as a Java developer at Attune Infocom Pvt. Ltd. He is totally focused on open source technologies, and he is very much interested in sharing his knowledge with the open source community.

About the Reviewers

Mark Kerzner holds degrees in Law, Mathematics, and Computer Science. He has been designing software for many years and Hadoop-based systems since 2008. He is the President of SHMsoft, a provider of Hadoop applications for various verticals. He is a co-founder of the Hadoop Illuminated training and consulting firm, and the co-author of the open source *Hadoop Illuminated* book. He has authored and co-authored a number of books and patents.

I would like to acknowledge the help of my colleagues, in particular Sujee Maniyam, and last but not least, my multitalented family.

Shriram Sridharan is a student at the University of Wisconsin-Madison, pursuing his Masters' degree in Computer Science. He is currently working in Prof. Jignesh Patel's research group. His current interests lie in the areas of databases and distributed systems. He received his Bachelor's degree from the College of Engineering Guindy, Anna University, Chennai and has two years of work experience. You can contact him at `shriramsc.wisc.edu`.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Apache Nutch is an open source web crawler software that is used for crawling websites. It is extensible and scalable. It provides facilities for parsing, indexing, and scoring filters for custom implementations. This book is designed for making you comfortable in applying web crawling and data mining in your existing application. It will demonstrate real-world problems and give the solutions to those problems with appropriate use cases.

This book will demonstrate all the practical implementations hands-on so readers can perform the examples on their own and make themselves comfortable. The book covers numerous practical implementations and also covers different types of integrations.

What this book covers

Chapter 1, *Getting Started with Apache Nutch*, covers the introduction of Apache Nutch, including its installation, and guides you for crawling, parsing, and creating plugins with Apache Nutch. By the end of this chapter, you will be able to install Apache Nutch in your own environment, and also be able to crawl and parse websites. Additionally, you will be able to create a Nutch plugin.

Chapter 2, *Deployment, Sharding, and AJAX Solr with Apache Nutch*, covers the deployment of Apache Nutch on a particular server; that is, Apache Tomcat and Jetty. It also covers how sharding can take place with Apache Nutch using Apache Solr as a search tool. By the end of this chapter, you will be able to deploy Apache Solr on a server that contains the data crawled by Apache Nutch and also be able to perform sharding using Apache Nutch and Apache Solr. You will also be able to integrate AJAX with your running Apache Solr instance.

Chapter 3, *Integrating Apache Nutch with Apache Hadoop and Eclipse*, covers integration of Apache Nutch with Apache Hadoop and also covers how we can integrate Apache Nutch with Eclipse. By the end of this chapter, you will be able to set up Apache Nutch running on Apache Hadoop in your own environment and also be able to perform crawling in Apache Nutch using Eclipse.

Chapter 4, *Apache Nutch with Gora, Accumulo, and MySQL*, covers the integration of Apache Nutch with Gora, Accumulo, and MySQL. By the end of this chapter, you will be able to integrate Apache Nutch with Apache Accumulo as well as with MySQL. After that, you can perform crawling using Apache Nutch on Apache Accumulo and also on MySQL. You can also get the results of your crawled pages on Accumulo as well as on MYSQL. You can integrate Apache Solr too, as we have discussed before, and get your crawled pages indexed onto Apache Solr.

What you need for this book

You will require the following software to be installed before starting with the book:

- Java 6 or higher; Apache Nutch requires JDK 6 or a later version. JDK 6 can be downloaded from `http://www.oracle.com/technetwork/java/javase/downloads/jdk6downloads-1902814.html`
- Apache Nutch 2.2.1, which can be downloaded from `http://nutch.apache.org/downloads.html`
- Apache Hbase 0.90.4, which can be downloaded from `http://archive.apache.org/dist/hbase/hbase 0.90.4/`
- Apache Solr 3.6.2, which can be downloaded from `http://archive.apache.org/dist/lucene/solr/`
- Apache Tomcat 7.0.41, which can be downloaded from `http://tomcat.apache.org`
- Apache Solr 4.3.0, which can be downloaded from `http://archive.apache.org/dist/lucene/solr/4.3.0/`
- Reuters.data.tar.gz, which can be downloaded from `https://github.com/downloads/evolvingweb/ajax-solr/reuters_data.tar.gz`
- Apache Hadoop, which can be downloaded from `http://www.apache.org/dyn/closer.cgi/hadoop/common/`
- Apache Nutch 1.4, which can be downloaded from `http://nutch.apache.org/downloads.html`
- Eclipse Juno, which can be downloaded from `http://www.eclipse.org/downloads/packages/release/juno/r`

- Subclipse, which can be downloaded from `http://subclipse.tigris.org/`
- IvyDE plugin, which can be downloaded from `http://ant.apache.org/ivy/ivyde/download.cgi`
- M2e plugin, which can be downloaded from `http://marketplace.eclipse.org/content/maven-integration-eclipse`
- Apache ZooKeeper, which can be downloaded from `http://zookeeper.apache.org/releases.html`
- Apache Accumulo, which can be downloaded from `http://accumulo.apache.org/downloads/`

Who this book is for

This book is for those who are looking to integrate web crawling and data mining into their existing applications as well as for the beginners who want to start with web crawling and data mining. It will provide complete solutions for real-time problems.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Go to the `solr` directory, which you will find in `/usr/local/SOLR_HOME`."

A block of code is set as follows:

```
<field name="id" type="string" indexed="true" stored="true"
required="true" multiValued="false" />
<field name="sku" type="text_en_splitting_tight" indexed="true"
stored="true" omitNorms="true"/>
```

Any command-line input or output is written as follows:

```
curl 'http://localhost:8983/solr/collection1/update' --data-binary
'<commit/>' -H 'Content-type:application/xml'
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Getting Started with Apache Nutch

Apache Nutch is a very robust and scalable tool for web crawling; it can be integrated with the scripting language Python for web crawling. You can use it whenever your application contains huge data and you want to apply crawling on your data.

This chapter covers the introduction to Apache Nutch and its installation, and also guides you on crawling, parsing, and creating plugins with Apache Nutch. It will start from the basics of how to install Apache Nutch and then will gradually take you to the crawling of a website and creating your own plugin.

In this chapter we will cover the following topics:

- Introducing Apache Nutch
- Installing and configuring Apache Nutch
- Verifying your Nutch installation
- Crawling your first website
- Setting up Apache Solr for search
- Integrating Solr with Nutch
- Crawling websites using crawl script
- Crawling the web, URL filters, and the CrawlDb
- Parsing and parsing filters
- Nutch plugins and Nutch plugin architecture

By the end of this chapter, you will be comfortable playing with Apache Nutch as you will be able to configure Apache Nutch yourself in your own environment and you will also have a clear understanding about how crawling and parsing take place with Apache Nutch. Additionally, you will be able to create your own Nutch plugin.

Introduction to Apache Nutch

Apache Nutch is open source WebCrawler software that is used for crawling websites. You can create your own search engine like Google, if you understand Apache Nutch clearly. It will provide you with your own search engine, which can increase your application page rank in searching and also customize your application searching according to your needs. It is extensible and scalable. It facilitates parsing, indexing, creating your own search engine, customizing search according to needs, scalability, robustness, and ScoringFilter for custom implementations. **ScoringFilter** is a Java class that is used while creating the Apache Nutch plugin. It is used for manipulating scoring variables.

We can run Apache Nutch on a single machine as well as on a distributed environment such as Apache Hadoop. It is written in Java. We can find broken links using Apache Nutch and create a copy of all the visited pages for searching over, for example, while building indexes. We can find web page hyperlinks in an automated manner.

Apache Nutch can be integrated with Apache Solr easily and we can index all the web pages that are crawled by Apache Nutch to Apache Solr. We can then use Apache Solr for searching the web pages which are indexed by Apache Nutch. Apache Solr is a search platform that is built on top of Apache Lucene. It can be used for searching any type of data, for example, web pages.

Installing and configuring Apache Nutch

In this section, we are going to cover the installation and configuration steps of Apache Nutch. So we will first start with the installation dependencies in Apache Nutch. After that, we will look at the steps for installing Apache Nutch. Finally, we will test Apache Nutch by applying crawling on it.

Installation dependencies

The dependencies are as follows:

- Apache Nutch 2.2.1
- HBase 0.90.4

- Ant
- JDK 1.6

Apache Nutch comes in different branches, for example, 1.x, 2.x, and so on. The key difference between Apache Nutch 1.x and Apache Nutch 2.x is that in the former, we have to manually type each command step-by-step for crawling, which will be explained later in this chapter. In the latter, Apache Nutch developers create a crawl script that will do crawling for us by just running that script; there is no need to type commands step-by-step.

There may be more differences but I have covered just one.

I have used Apache Nutch 2.2.1 because it is the latest version at the time of writing this book. The steps for installation and configuration of Apache Nutch are as follows:

1. Download Apache Nutch from the Apache website. You may download Nutch from `http://nutch.apache.org/downloads.html`.
2. Click on **apache-nutch-2.2.1-src.tar.gz** under the **Mirrors** column in the **Downloads** tab. You can extract it by typing the following commands:

   ```
   #cd $NUTCH_HOME
   # tar -zxvf apache-nutch.2.2.1-src.tar.gz
   ```

   Here `$NUTCH_HOME` is the directory where your Apache Nutch resides.

3. Download HBase. You can get it from `http://archive.apache.org/dist/hbase/hbase-0.90.4/`.

   HBase is the Apache Hadoop database that is distributed, a big data store, scalable, and is used for storing large amounts of data. You should use Apache HBase when you want real-time read/write accessibility of your big data. It provides modular and linear scalability. Read and write operations are very consistent. Here, we will use Apache HBase for storing data, which is crawled by Apache Nutch. Then we can log in to our database and access it according to our needs.

4. We now need to extract HBase, for example, `Hbase.x.x.tar.gz`. Go to the terminal and reach up to the path where your `Hbase.x.x.tar.gz` resides. Then type the following command for extracting it:

   ```
   tar -zxvf Hbase.x.x.tar.gz
   ```

   It will extract all the files in the respective folder.

5. Now we need to do HBase configuration. First, go to `hbase-site.xml`, which you will find in `<Your HBase home>/conf` and modify it as follows:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>hbase.rootdir</name>
<value><Your path></value>
<!— You need to create one directory and assign a path up to that
directory. That directory will be used by Apache Hbase to store
all relevant information. à</property>
<property>
<name>hbase.zookeeper.property.dataDir</name>
<value><Your path></value>
<!— You need to create one directory and assign a path up to
that directory. That directory will be used by Apache Hbase
to store all relevant information related to Apache zookeeper
which comes inbuilt with Apache Hbase. Apache Zookeeper is an
open source server which is used for distributed coordination.
You can learn more about Apache Zookeeper from
https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index
</property>
</configuration>
```

   Just make sure that the `hosts` file under `etc` contains the loop back address, which is `127.0.0.1` (in some Linux distributions, it might be `127.0.1.1`). Otherwise you might face an issue while running Apache HBase.

6. Specify Gora backend in `nutch-site.xml`. You will find this file at `$NUTCH_HOME/conf`.

```
<property>
<name>storage.data.store.class</name>
<value>org.apache.gora.hbase.store.HBaseStore</value>
<description>Default class for storing data</description>
</property>
```

   The explanation of the preceding configuration is as follows:

   - Find the name of the data store class for storing data of Apache Nutch:

     ```
     <name>storage.data.store.class</name>
     ```

   - Find the database in which all the data related to HBase will reside:

     ```
     <value>org.apache.gora.hbase.store.HBaseStore</value>
     ```

7. Make sure that the `HBasegora-hbase` dependency is available in `ivy.xml`. You will find this file in `<Your Apache Nutch home>/ivy`. Put the following configuration into the `ivy.xml` file:

```
<!-- Uncomment this to use HBase as Gora backend. -->
<dependency org="org.apache.gora" name="gora-hbase" rev="0.2"
conf="*-
>default" />
```

   The last line would be commented by default. So you need to uncomment it.

8. Make sure that `HBaseStore` is set as the default data store in the `gora.properties` file. You will find this file in `<Your Apache Nutch home>/conf`. Put the following configuration into `gora.properties`:

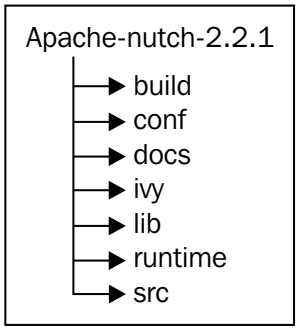
```
gora.datastore.default=org.apache.gora.hbase.store.HBaseStore
```

   The preceding line would be commented by default. So uncomment it.

9. Go to Apache Nutch home directory. This directory would be `<Your Apache Nutch home directory>`. Go there and type the following command from your terminal:

```
ant runtime
```

   This will build your Apache Nutch and create the respective directories in the Apache Nutch's home directory. It is needed because Apache Nutch 2.x is only distributed as source code. The Apache Nutch 1.x branch is distributed as binary. So in that, this stage is not required. The tree structure of the generated directories would be as shown in the following diagram:



The preceding diagram shows the directory structure of Apache Nutch, which we built in the preceding step. The `runtime` and `build` directories will be newly generated after building `apache-nutch-2.2.1`. The rest of the directories already exist in `apache-nutch-2.2.1`. The following directories are listed:

- The `build` directory contains all the required JAR files that Apache Nutch has downloaded at the time of building
- The `conf` directory contains all the configuration files which are required for crawling
- The `docs` directory contains the documentation that will help the user to perform crawling
- The `ivy` directory contains the required configuration files in which the user needs to add certain configurations for crawling
- The `runtime` directory contains all the necessary scripts which are required for crawling
- The `src` directory contains all the Java classes on which Apache Nutch has been built

Ant is the tool which is used for building your project and which will resolve all the dependencies of your project. It will fetch the required JAR files from the Internet by running the `build.xml` file that is required for running Ant. You need to define all the dependencies in `build.xml`. So when you type `ant` at runtime, it will search for the `build.xml` file in the directory from where you have hit this command, and once found, it will fetch all the required JAR files that you have mentioned in `build.xml`. You have to install Ant if it is not installed already. You can refer to `http://www.ubuntugeek.com/how-to-install-ant-1-8-2-using-ppa-on-ubuntu.html` for a guide to the installation of Ant.

10. Make sure HBase is started and is working properly. To check whether HBase is running properly, go to the home directory of Hbase. Type the following command from your terminal:

    ```
    ./bin/hbase shell
    ```

    If everything goes well, you will get an output as follows:

    ```
    HBase Shell; enter 'help<RETURN>' for list of supported commands.
    Type "exit<RETURN>" to leave the HBase Shell
    Version: 0.90.4, r1001068, Fri Sep 24 13:55:42 PDT 2010
    ```

11. This completes your installation of Apache Nutch. Now you should be able to use it by going to the `bin` directory of Apache Nutch. You will find this directory at `<Your Apache Nutch home>/runtime/local`.

The local directory contains all the configuration files which are required to perform crawling. The script for crawling also resides inside this directory. The `runtime` directory contains the `local` directory and the `deploy` directory. You should find more details in the logs at `<Your Apache Nutch home>/runtime/local/logs/hadoop.log`.

Verifying your Apache Nutch installation

Once Apache Nutch is installed, it is important to check whether it is working up to the mark or not. For this, a verification process is required. The steps for verifying Apache Nutch installation are as follows:

1. Go to the `local` directory of Apache Nutch from your terminal. You will find this directory at `<Your Apache Nutch home directory>/runtime`. Type the following command here:

   ```
   bin/nutch
   ```

   If everything is successful, you will get the output as follows:

   ```
   Usage: nutch COMMAND
   ..
   ..
   ..
   Most commands will print help when invoked w/o parameters.
   ```

2. Run the following command if you see a `Permission denied` message:

   ```
   chmod +x bin/nutch
   ```

3. Set up `JAVA_HOME` if it's not set already. On your Mac system, you can run the following command or add it to `~/.bashrc`. You can open `~/.bashrc` by going to the root directory from your terminal and typing `gedit~/.bashrc`.

   ```
   export JAVA_HOME=<Your Java path>
   ```

Crawling your first website

We have now completed the installation of Apache Nutch. It's now time to move to the key section of Apache Nutch, which is crawling. Crawling is driven by the Apache Nutch crawling tool and certain related tools for building and maintaining several data structures. It includes web database, the index, and a set of segments. Once Apache Nutch has indexed the web pages to Apache Solr, you can search for the required web page(s) in Apache Solr. The steps for crawling are as follows:

1. Add your agent name in the `value` field of the `http.agent.name` property in the `nutch-site.xml` file. The `nutch-site.xml` file is the configuration file from where Apache Nutch will fetch the necessary details at the time of crawling. We will define different properties in this file, as you will see in the following code snippet. You will find this file located at `<Your Apache Nutch home>/runtime/local/conf`. Add the following configuration into `nutch-site.xml`:

   ```
   <configuration>
   <property>
   <name>http.agent.name</name>
   <value>My Nutch Spider</value>
   </property>
   </configuration>
   ```

   The explanation of the preceding configuration is as follows:

   - Find HTTP agent name as follows:

     ```
     <name>http.agent.name</name>
     ```

   - Find HTTP agent value as follows. You can specify any value here. Apache Nutch requires this value while crawling the website.

     ```
     <value>My Nutch Spider</value>
     ```

2. Go to the `local` directory of Apache Nutch. You will find this directory located at `<Your Apache Nutch home>/runtime`. Create a directory called `urls` inside it by following these steps:

   a. Find the command for creating the `urls` directory as follows:

      ```
      #mkdir -p urls
      ```

   b. If you are a Windows user, the following command should be used:

      ```
      #mkdir urls
      ```

   c. The following command will take you inside the `urls` directory:

      ```
      #cd urls
      ```

3. Now create the `seed.txt` file inside the `urls` directory and put the following content:

   ```
   http://nutch.apache.org/
   ```

4. This is the URL which is used for crawling. You can put *n* number of URLs but one URL per line. The format of the URL would be `http://<Respective url>`. You can comment by putting `#` at the start of the line. An example would be as follows:

   ```
   # Your commented text is here.
   ```

5. Edit the `regex-urlfilter.txt` file. This file is used for filtering URLs for crawling. So whenever crawling is performed for the URL, Apache Nutch will match the respective URL that we are putting inside `seed.txt`, with the pattern defined in this file for that URL and crawl accordingly. As you will see shortly, we have applied crawling on `http://nutch.apache.org` and we have set the pattern inside this file. So it will tell Apache Nutch that all the URLs which end up with `nutch.apache.org` need to be crawled. You will find this file located at `<Your Apache Nutch home>/conf`; replace the following lines with a regular expression matching the domain you wish to crawl:

   ```
   # accept anything else
   +.
   ```

6. For example, if you wish to limit the crawl to the `nutch.apache.org` domain, this line should read as follows:

   ```
   +^http://([a-z0-9]*\.)*nutch.apache.org/
   ```

Installing Apache Solr

Apache Solr is a search platform which is built on top of Apache Lucene. It can be used for searching any type of data, for example, web pages. It's a very powerful searching mechanism and provides full-text search, dynamic clustering, database integration, rich document handling, and much more. Apache Solr will be used for indexing URLs which are crawled by Apache Nutch and then one can search the details in Apache Solr crawled by Apache Nutch. Follow these steps for installation of Apache Solr:

1. Download Apache Solr from `http://archive.apache.org/dist/lucene/solr/`.

2. Extract it by typing the following commands:

```
cd /usr/local
$ sudo tar xzf apache-solr-3.6.2/
$ sudo mv apache-solr-3.6.2/ solr
```

   This will extract all the files of Apache Solr in the respective folder.

3. Now we need to set the path of the `JAVA_HOME` variable in the `~/.bashrc` file. To open this file, go to the root directory from your terminal and type the following command:

```
gedit ~/.bashrc
```

   Put the following configuration into the `~/.bashrc` file:

```
 #Set SOLR home
export SOLR_HOME=/usr/local/solr/example/solr
```

   This creates an environment variable called `SOLR_HOME`. This classpath variable is required for Apache Solr to run. When you start Apache Solr, it will search for this variable inside your `.bashrc` file for locating your Apache Solr and it will give an error if something goes wrong in the configuration.

4. Go to the example directory from your terminal. You will find this directory located in your Apache Solr's home directory. Type the following command to start Apache SOLR:

```
java -jar start.jar
```

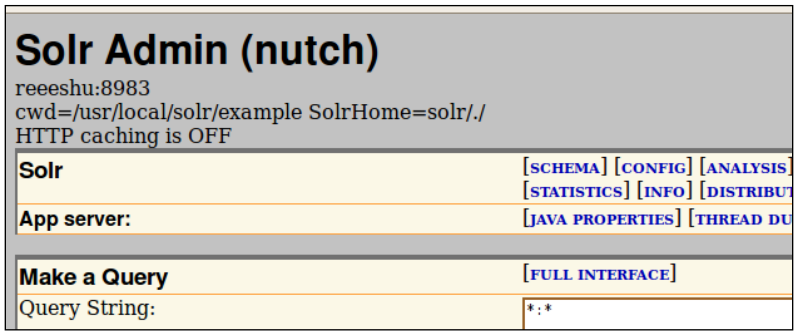
   If all succeeds, you will get following output:

```
INFO: solr home defaulted to 'solr/' (could not find system property or JNDI)
23 Nov, 2013 4:25:24 AM org.apache.solr.servlet.SolrUpdateServlet init
INFO: SolrUpdateServlet.init() done
2013-11-23 04:25:24.762:INFO::Started SocketConnector@0.0.0.0:8983
```

5. Verify Apache Solr installation by hitting the following URL on your browser:

```
http://localhost:8983/solr/admin/
```

   You will get the image of Running Apache Solr on your browser, as shown in the following screenshot:



Integration of Solr with Nutch

In the above steps, we have installed Apache Nutch and Apache Solr correctly. Integration is required for indexing URLS to Apache Solr crawled by Apache Nutch. So once Apache Nutch finishes with crawling and indexing URLs to Apache Solr, you can search for particular documents on Apache Solr and get the expected results. The steps for integrating Apache Solr with Apache Nutch are as follows:

1. Copy the `schema.xml` file. You will find this file at `<Your Apache Nutch home>/conf`. Put it into the `conf` directory of Apache SOLR. You will find this directory in your Apache Solr's home directory. Enter the following command:

   ```
   cp<Respective Directory where Apache Nutch Resides>/conf/schema.
   xml <Respective Directory where Apache SOLR resides>/example/solr/
   conf/
   ```

2. Go to the `example` directory. You will find this directory in your Apache Solr's home directory. Type the following command to restart Apache SOLR:

   ```
   java-jar start.jar
   ```

Crawling your website using the crawl script

Apache Nutch 2.2.1 comes with the crawl script facility which does crawling by just executing one single script. In the earlier version, we had to manually perform the steps of generating data, fetching data, parsing data, and so on for performing crawling. I have installed both the Apache Nutch and Apache Solr setups correctly. The steps for crawling your website using the crawl script are as follows:

1. Go to the home directory of HBase from your terminal. You will find this directory located at the same location where your HBase resides. Start HBase by typing the following command:

   ```
   ./bin/start-hbase.sh
   ```

   If all succeeds, you will get the following output:

   ```
   Starting Master, logging on to logs/hbase-user-master-example.org.
   out.
   ```

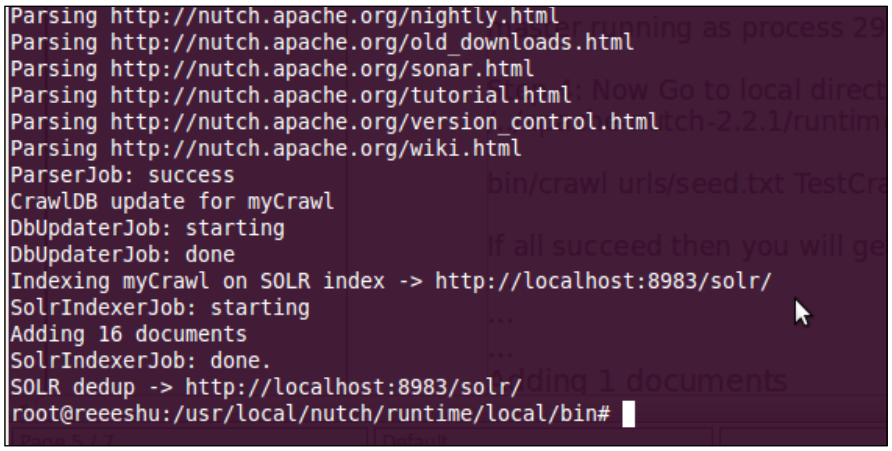
2. If you get the following output, it means HBase is already started. No need to start it.

   ```
   master running as process 2948. Stop it first.
   ```

3. Now go to the `local` directory of Apache Nutch from your terminal and perform some operations by typing the following command. You will find the `local` directory in `<Your Apache Nutch home>/runtime`.

   ```
   cd<Respective directory where Apache Nutch resides>/runtime
   bin/crawl urls/seed.txt TestCrawl http://localhost:8983/solr/2
   ```

   If all succeeds, you will get the following output:



   The command is explained as follows:

   - `urls/seed.txt`: `seed.txt` is the file which contains urls for crawling.
   - `TestCrawl`: This is the crawl data directory which will be automatically created inside Apache Hbase with the name `TestCrawl_Webpage`, which will contain information on all the URLs which are crawled by Apache Nutch.

- `http://localhost:8983/solr/`: This is the URL of running Apache Solr.
- `2`: This is the number of iterations, which will tell Apache Nutch in how many iterations this crawling will end.

You can modify the parameters according to your requirements. The crawl script has a lot of parameters to be set; it would be good to understand the parameters before setting up big crawls. Because you can use these parameters according to your requirements, you have to first study these parameters and then apply them.

Crawling the Web, the CrawlDb, and URL filters

Crawling the Web is already explained above. You can add more URLs in the `seed.txt` file and crawl the same.

When a user invokes a crawling command in Apache Nutch 1.x, CrawlDB is generated by Apache Nutch which is nothing but a directory and which contains details about crawling. In Apache 2.x, CrawlDB is not present. Instead, Apache Nutch keeps all the crawling data directly in the database. In our case, we have used Apache HBase, so all crawling data would go inside Apache HBase. The following are details of how each function of crawling works.

A crawling cycle has four steps, in which each is implemented as a Hadoop MapReduce job:

- `GeneratorJob`
- `FetcherJob`
- `ParserJob` (optionally done while fetching using 'fetch.parse')
- `DbUpdaterJob`

Additionally, the following processes need to be understood:

- `InjectorJob`
- `Invertlinks`
- Indexing with Apache Solr

First of all, the job of an Injector is to populate initial rows for the web table. The `InjectorJob` will initialize `crawldb` with the URLs that we have provided. We need to run the `InjectorJob` by providing certain URLs, which will then be inserted into `crawlDB`.

Then the `GeneratorJob` will use these injected URLs and perform the operation. The table which is used for input and output for these jobs is called `webpage`, in which every row is a URL (web page). The row key is stored as a URL with reversed host components so that URLs from the same TLD and domain can be kept together and form a group. In most NoSQL stores, row keys are sorted and give an advantage. Using specific rowkey filtering, scanning will be faster over a subset, rather than scanning over the entire table.

Following are the examples of rowkey listing:

- `org.apache..www:http/`
- `org.apache.gora:http/`

Let's define each step in depth so that we can understand crawling step-by-step.

Apache Nutch contains three main directories, `crawlDB`, `linkdb`, and a set of segments. `crawlDB` is the directory which contains information about every URL that is known to Apache Nutch. If it is fetched, `crawlDB` contains the details when it was fetched. The `linkdatabase` or `linkdb` contains all the links to each URL which will include source URL and also the anchor text of the link. A set of segments is a URL set, which is fetched as a unit. This directory will contain the following subdirectories:

- A `crawl_generate` job will be used for a set of URLs to be fetched
- A `crawl_fetch` job will contain the status of fetching each URL
- A content will contain the content of rows retrieved from every URL

Now let's understand each job of crawling in detail.

InjectorJob

The Injector will add the necessary URLs to the `crawlDB`. `crawlDB` is the directory which is created by Apache Nutch for storing data related to crawling. You need to provide URLs to the `InjectorJob` either by downloading URLs from the Internet or by writing your own file which contains URLs. Let's say you have created one directory called `urls` that contains all the URLs needed to be injected in `crawlDB`; the following command will be used for performing the `InjectorJob`:

```
#bin/nutch inject crawl/crawldb urls
```

`urls` will be the directory which contains all the URLs that are needed to be injected in `crawlDB`. `crawl/crawldb` is the directory in which injected URLs will be placed. After performing this job, you will have a number of unfetched URLs inside your database (`crawlDB`).

GeneratorJob

Once we are done with the `InjectorJob`, it's time to fetch the injected URLs from CrawlDB. So for fetching the URLs, you need to perform the `GeneratorJob` first. The following command will be used for `GeneratorJob`:

```
#bin/nutch generate crawl/crawldb crawl/segments
```

`crawldb` is the directory from where URLs are generated. `segments` is the directory which is used by the `GeneratorJob` to fetch the necessary information required for crawling.

FetcherJob

The job of the fetcher is to fetch the URLs which are generated by the `GeneratorJob`. It will use the input provided by `GeneratorJob`. The following command will be used for the `FetcherJob`:

```
#bin/nutch fetch -all
```

Here I have provided input parameters—this means that this job will fetch all the URLs that are generated by the `GeneratorJob`. You can use different input parameters according to your needs.

ParserJob

After the `FetcherJob`, the `ParserJob` is to parse the URLs that are fetched by `FetcherJob`. The following command will be used for the `ParserJob`:

```
# bin/nutch parse -all
```

I have used input parameters—all of which will parse all the URLs fetched by the `FetcherJob`. You can use different input parameters according to your needs.

DbUpdaterJob

Once the `ParserJob` has completed its task, we need to update the database by providing results of the `FetcherJob`. This will update the respective databases with the last fetched URLs. The following command will be used for the `DbUpdaterJob`:

```
# bin/nutch updatedb crawl/crawldb -all
```

I have provided input parameters, all of which will update all the URLs that are fetched by the `FetcherJob`. You can use different input parameters according to your needs. After performing this job, the database will contain updated entries of all the initial pages and the new entities which correspond to the newly discovered pages that are linked from the initial set.

Invertlinks

Before applying indexing, we need to first invert all the links. After this we will be able to index incoming anchor text with the pages. The following command will be used for `Invertlinks`:

```
# bin/nutch invertlinks crawl/linkdb -dir crawl/segments
```

Indexing with Apache Solr

At the end, once crawling has been performed by Apache Nutch, you can index all the URLs that are crawled by Apache Nutch to Apache Solr, and after that you can search for the particular URL on Apache Solr. The following command will be used for indexing with Apache Solr:

```
#bin/nutch solrindex http://127.0.0.1:8983/solr/ crawl/crawldb -linkdb
crawl/linkdb crawl/segments/*
```

Parsing and parse filters

Parsing is a task or process by which a parse object is created and populated within the data. Parsing contains the parsed text of each URL, the outlink URLs used to update `crawldb`, and outlinks and metadata parsed from each URL. Parsing is also done by crawl script, as explained earlier; to do it manually, you need to first execute inject, generate, and fetch commands, respectively:

For generating, the following command will be used:

```
bin/nutch generate -topN 1
```

For fetching, the following command will be used:

```
bin/nutch fetch -all
```

For parsing, the following command will be used:

```
bin/nutch parse -all
```

The preceding commands are the individual commands required for parsing. To perform these, go to the Apache Nutch home directory. This directory would be where Apache Nutch resides. Then type the following commands:

For generating records, the following command will be used:

```
bin/nutch generate -topN 1
```

For fetching records, the following command will be used:

```
bin/nutch fetch all
```

For parsing records, the following command will be used:

```
bin/nutch parse -all
```

The `HtmlParseFilter` permits one to add additional metadata to HTML parses.

Webgraph

Webgraph is a component which is used for creating databases. It will create databases for inlinks, outlinks, and nodes that are used for holding the number of outlinks and inlinks to a URL and for the current score of the URL. You need to go for Apache Nutch 1.x to use this, as Apache 2.x will not work. Webgraph will run once all the segments are fetched and ready to be processed. It can be found at `org.apache.nutch.scoring.webgraph.WebGraph`. Just go to the `bin` directory, which you will find in `$NUTCH_HOME/runtime/local`, and fire the following command:

```
#bin/nutch webgraph (-segment <segment> | -segmentDir <segmentDir> |
-webgraphdb <webgraphdb>) [-filter -normalize] | -help
```

If you only type `#bin/nutch webgraph`, it will show you the usage as follows:

```
usage: WebGraph
-help
show this help message
-segment <segment>
the segment(s) to use
-webgraphdb<webgraphdb> the web graph database to use
```

Loops

Loops is used for determining spam sites by determining link cycles in a Webgraph. So once the Webgraph is completed, we can start the process of link analysis. An example of a link cycle is, P is linked to Q, Q is linked R, R is linked S, and then again S is linked to P. Due to its large expense and time and space requirements, it cannot be run on more than four levels. Its benefit to cost ratio is very low. It helps the LinkRank program to identify spam sites, which can then be discounted in later LinkRank programs. There can be another way to perform this function with a different algorithm. It is just placed here for the purpose of completeness. Its usage in the current large Webgraph is discouraged. Loops can be found at `org.apache.nutch.scoring.webgraph.Loops`. The usage details of loops are as follows:

```
usage: Loops
-help
show this help message
-webgraphdb<webgraphdb> the web graph database to use
```

LinkRank

LinkRank is used for performing an iterative link analysis. It is used for converging to stable global scoring for each URL. It starts with a common scoring for each URL like PageRank. It creates a global score for each and every URL based on the number of incoming links and also scores for those links and total outgoing links from the page. It is an iterative process and scores converge after a given number of iterations. It differs from PageRank in the way that links internal to a website and reciprocal links in between websites could be ignored. One can configure iterations too. The default number of iterations to be performed is 10. Unlike the OPIC scoring, the LinkRank program doesn't track scores from one processing time to another. Both Webgraph and link scores are recreated on each processing run. So we do not have any problems in increasing scores. LinkRank wants the Webgraph program to be completed successfully and stores the output scoring of each URL in the node database of the Webgraph. LinkRank is found at `org.apache.nutch.scoring.webgraph.LinkRank`. A printout of the program's usage is as follows:

```
usage: LinkRank
-help show this help message
-webgraphdb<webgraphdb> the web graph db to use
```

ScoreUpdater

After completing the LinkRank program and link analysis, your scores must be updated inside the crawl database working with the current Apache Nutch functionality. The `ScoreUpdater` program stores the scores in the `node` database of the Webgraph and updates them inside `crawldb`.

```
usage: ScoreUpdater
 -crawldb<crawldb>          the crawldb to use
 -help                       show this help message
 -webgraphdb<webgraphdb>    the webgraphdb to use
```

A scoring example

This example runs the new scoring and indexing systems from start to end. The new scoring functionality can be found at `org.apache.nutch.scoring.webgraph`. The package contains multiple programs that will build web graphs, performing a stable convergent link analysis, and updating `crawldb` with those scores. For doing scoring, go to the local directory from the terminal. You will find this directory in `<Respective directory where Apache Nutch resides>/runtime` and type the following commands:

```
bin/nutch inject crawl/crawldb crawl/urls/
bin/nutch generate crawl/crawldb/ crawl/segments
bin/nutch fetch crawl/segments/20090306093949/
bin/nutchupdatedb crawl/crawldb/ crawl/segments/20090306093949/
bin/nutchorg.apache.nutch.scoring.webgraph.WebGraph -segment
crawl/segments/20090306093949/ -webgraphdb crawl/webgraphdb
```

Apache Nutch 2.2.1 does not support this. So I have configured it with `apache-nutch-1.7`. You can install `apache-nutch-1.7` in the same way as `apache-nutch-2.2.1`.

Webgraph will be used on larger web crawls to create web graphs. The following options are interchangeable with their corresponding configuration options:

```
<!--linkrank scoring properties -->
<property>
```

```
<name>link.ignore.internal.host</name>
<value>true</value>
<description>Ignore outlinks to the same hostname.</description>
</property>
<property>
<name>link.ignore.internal.domain</name>
<value>true</value>
<description>Ignore outlinks to the same domain.</description>
</property>

<property>
<name>link.ignore.limit.page</name>
<value>true</value>
<description>Limit to only a single outlink to the same page.</
description>
</property>

<property>
<name>link.ignore.limit.domain</name>
<value>true</value>
<description>Limit to only a single outlink to the same domain.</
description>
</property>
```

But by default, if you are doing only crawling of pages inside a domain or inside a set of subdomains, all the outlinks will be ignored and you come up having an empty Webgraph. Type the following commands:

```
bin/nutchorg.apache.nutch.scoring.webgraph.Loops -webgraphdb crawl/
webgraphdb/

bin/nutchorg.apache.nutch.scoring.webgraph.LinkRank -webgraphdb crawl/
webgraphdb/

bin/nutchorg.apache.nutch.scoring.webgraph.ScoreUpdater -crawldb crawl/
crawldb -webgraphdb crawl/webgraphdb/

bin/nutchorg.apache.nutch.scoring.webgraph.NodeDumper -scores -topn 1000
-webgraphdb crawl/webgraphdb/ -output crawl/webgraphdb/dump/scores

bin/nutchreaddb crawl/crawldb/ -stats
```

Use the `GenericOptionsParser` for parsing the arguments. Applications should implement tools to process these.

The command shown in the following screenshot will be used for showing statistics for `CrawlDbcrawl/crawldb/`:

```
reeeshu@reeeshu:~$ sudo -s
root@reeeshu:~# cd /usr/local/nutch/runtime/local/bin/
root@reeeshu:/usr/local/nutch/runtime/local/bin# ./nutch readdb myCrawl/
CrawlDb statistics start: myCrawl/crawldb/
Statistics for CrawlDb: myCrawl/crawldb/
TOTAL urls:     457
retry 0:        457
min score:      0.0
avg score:      0.0
max score:      0.0
status 1 (db_unfetched):        421
status 6 (db_notmodified):      36
CrawlDb statistics: done
root@reeeshu:/usr/local/nutch/runtime/local/bin#
```

The Apache Nutch plugin

The plugin system displays how Nutch works and allows us to customize Apache Nutch to our personal needs in a very flexible and maintainable manner. Everyone wants to use Apache Nutch for crawling websites. But writing an own plugin will be a challenge at one point or another. There are many changes in the `nutch-site.xml` and `schema.xml` files stored at `apache-nutch-2.2.1/conf/`. But simply imagine you would like to add a new field to the index by doing some custom analysis of a parsed web page content to Solr as an additional field.

The Apache Nutch plugin example

This example will focus on the `urlmeta` plugin. In this example we will use Apache Nutch 1.7. Its aim is to provide a comprehensive knowledge of the Apache Nutch plugin.

This example covers the integral components required to develop and use a plugin. As you can see, inside the `plugin` directory located at `$NUTCH_HOME/src/`, the folder `urlmeta` contains the following:

- A `plugin.xml` file that tells Nutch about your plugin
- A `build.xml` file that tells Ant how to build your plugin
- An `ivy.xml` file containing either the description of the dependencies of a module, its published artifacts and its configurations, or else the location of another file which specifies this information

- A `/src` directory containing the source code of our plugin with the directory structure is shown in the hierarchical view, as follows:
    - `plugin.xml`
    - `build.xml`
    - `ivy.xml`
    - `src/java/org/apache/nutch/indexer/urlmeta/$`
    - `package.html`
    - `URLMetaIndexingFilter.java`
    - `src/java/org/apache/nutch/indexer/urlmeta/$`
    - `Scoring`
    - `package.html`
    - `URLMetaScoringFilter.java`

Now we need to configure the `plugin.xml`, `build.xml`, and `ivy.xml` files.

Modifying plugin.xml

Your `plugin.xml` file should look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
id="urlmeta"
name="URL Meta Indexing Filter"
version="1.0.0"
provider-name="sgonyea">
```

The preceding code defines ID, name, version, and provider name of your plugin.

```
<runtime>
<library name="urlmeta.jar">
<export name="*"/>
</library>
</runtime>
```

The preceding code defines the library, which is a JAR file and export command of your plugin.

```
<requires>
<import plugin="nutch-extensionpoints"/>
</requires>
```

The preceding code is used for importing nutch-extension points for your plugin.

```
<extension id="org.apache.nutch.indexer.urlmeta"
name="URL Meta Indexing Filter"
point="org.apache.nutch.indexer.IndexingFilter">
<implementation id="indexer-urlmeta"
                    class="org.apache.nutch.indexer.urlmeta.
URLMetaIndexingFilter"/>
</extension>
```

The preceding code is used for defining extension ID, extension name, extension point, implementation ID, and implementation class for your plugin.

```
<extension id="org.apache.nutch.scoring.urlmeta"
name="URL Meta Scoring Filter"
point="org.apache.nutch.scoring.ScoringFilter">
<implementation id="scoring-urlmeta"
class="org.apache.nutch.scoring.urlmeta.URLMetaScoringFilter" />
</extension>
</plugin>
```

The preceding configuration is written to tell Apache Nutch about your plugin and register your plugin with Apache Nutch.

In its simplest form, the preceding configuration looks as follows:

```
<?xml version="1.0"?>

<project name="recommended" default="jar-core">

<import file="../build-plugin.xml"/>
</project>
```

The preceding code will be used for building your plugin using Ant.

Describing dependencies with the ivy module

The `ivy.xml` file is used to describe the dependencies of the plugin on other libraries:

```
<ivy-module version="1.0">
<info organisation="org.apache.nutch" module="${ant.project.name}">
<license name="Apache 2.0"/>
<ivyauthor name="Apache Nutch Team" url="http://nutch.apache.org"/>
<description>
        Apache Nutch
```

```
</description>
</info>

<configurations>
<include file="${nutch.root}/ivy/ivy-configurations.xml"/>
</configurations>

<publications>
<!--get the artifact from our module name-->
<artifact conf="master"/>
</publications>

<dependencies>
</dependencies>

</ivy-module>
```

The preceding configuration contains either the description of the dependencies of a module, its published artifacts and its configurations, or else the location of another file which specifies this information.

The Indexer extension program

This is the source code for the `URLMetaIndexingFilter`. `IndexingFilter` is the extension point and an interface for adding metadata into the search index.

```
packageorg.apache.nutch.indexer.urlmeta;
importorg.apache.commons.logging.Log;
importorg.apache.commons.logging.LogFactory;
importorg.apache.hadoop.conf.Configuration;
importorg.apache.hadoop.io.Text;
importorg.apache.nutch.crawl.CrawlDatum;
importorg.apache.nutch.crawl.Inlinks;
importorg.apache.nutch.indexer.IndexingException;
importorg.apache.nutch.indexer.IndexingFilter;
importorg.apache.nutch.indexer.NutchDocument;
importorg.apache.nutch.parse.Parse;

public class URLMetaIndexingFilter implements IndexingFilter {

private static final Log LOG = LogFactory
                        .getLog(URLMetaIndexingFilter.class);
private static final String CONF_PROPERTY = "urlmeta.tags";
private static String[] urlMetaTags;
private Configuration conf;
```

```
        /**
         * This will take the metatags that you have listed in your
"urlmeta.tags"
         * property, and looks for them inside the CrawlDatum object.
If they exist,
         * this will add it as an attribute inside the NutchDocument.
         *
         * @see IndexingFilter#filter
         */
publicNutchDocument filter(NutchDocument doc, Parse parse, Text url,
CrawlDatum datum, Inlinksinlinks) throws IndexingException {
if (conf != null)
this.setConf(conf);

if (urlMetaTags == null || doc == null)
return doc;

for (String metatag : urlMetaTags) {
                        Text metadata = (Text) datum.getMetaData().
get(new Text(metatag));

if (metadata != null)
doc.add(metatag, metadata.toString());
               }

return doc;
        }

        /** Boilerplate */
public Configuration getConf() {
returnconf;
        }

        /**
         * handles conf assignment and pulls the value assignment from
the
         * "urlmeta.tags" property
         */
public void setConf(Configuration conf) {
this.conf = conf;

if (conf == null)
return;

urlMetaTags = conf.getStrings(CONF_PROPERTY);
        }
}
```

The Scoring extension program

This is the source code for the `URLMetaScoringFilter` extension. If the document being indexed has a recommended metatag, this extension adds a Lucene text field to the index called `recommended` with the content of that metatag.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Using your plugin with Apache Nutch

So the plugin has already been created. Now it's time to make it active. For that you need to make certain configurations with Apache Nutch. It will configure your plugin with Apache Nutch and after that you are able to use it as and when required. For that you need to edit your `nutch-site.xml` file by in putting the following content, which you will find in `$NUTCH_HOME/conf`.

```
<property>
<name>plugin.includes</name>
<value>protocol-http|urlfilter-regex|parse-(html|tika)|index-
(basic|anchor)|scoring-opic|urlnormalizer-(pass|regex|basic)|urlmeta</
value>
<description>
As you can see above, I have added urlmeta. Same way you can create
Apache Nutch plugin according to your needs. Regular expression naming
plugin directory names to
include.  Any plugin not matching this expression is excluded.
  In any case you need at least include the nutch-extensionpoints
plugin. By
defaultNutch includes crawling just HTML and plain text via HTTP,
and basic indexing and search plugins.
</description>
</property>
```

Compiling your plugin

And the last step of creating the Apache Nutch plugin is to compile your plugin. Once it is compiled, your newly created plugin will be indexed to Apache Solr and then you can search for the field, which is added as your plugin on Apache Solr; Apache Solr will give you the result as a separate document. So you need to compile and deploy your plugin by Ant. For that, you need to edit `build.xml` by in putting the following content. You will find this file in `$NUTCH_HOME/src/plugin`.

```
<ant dir="urlmeta" target="deploy" />
```

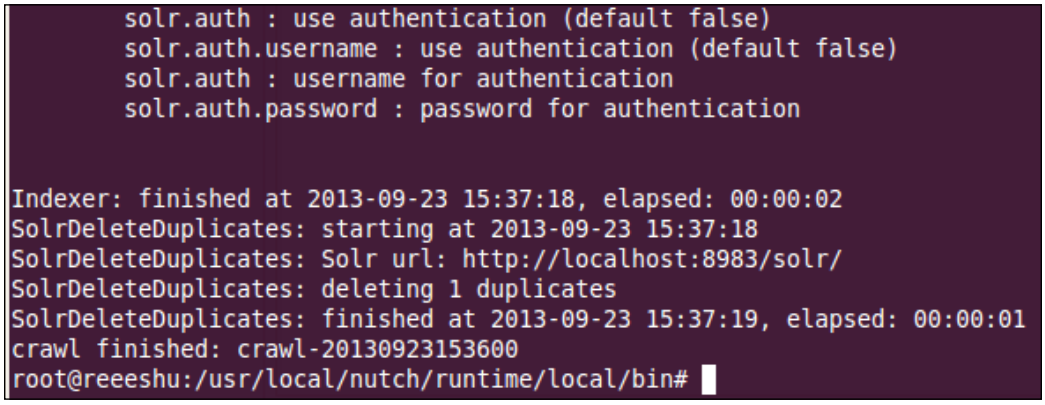
Now run the Ant command in the `$NUTCH_HOME` directory. It will get everything compiled and jarred up. Then execute the following command for crawling:

```
bin/nutch crawl ./urls/seed.txt/ -solr http://localhost:8983/solr/ -depth
3 -topN
```

Both scoring and indexing extensions will be used, which will enable us to search for metatags within our Solr index. You can go to Apache Solr on the browser by typing `localhost:<PORT>` and do an all query search. You will get your plugin indexed into the Apache Solr as follows:

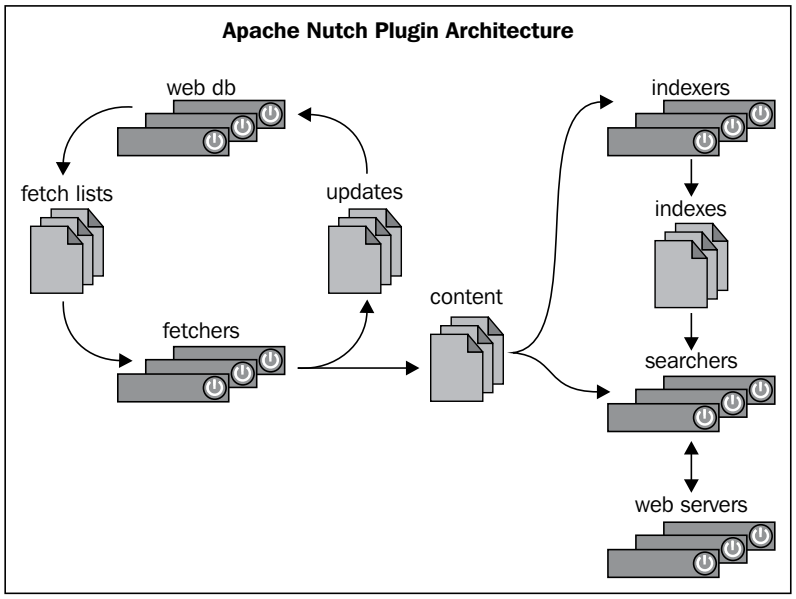
```
<doc>
  ...
  ...
  ...
</doc>
```

The following screenshot shows that the plugin has been successfully created:



Understanding the Nutch Plugin architecture

The following diagram illustrates the Apache Nutch plugin architecture which will show you how Apache Nutch functions. It will elaborate on searcher, indexer, web DB, and fetcher. It will provide you with the flow, how data is flowing from one stage to the next. It's a very key component of Apache Nutch. Therefore, understanding this architecture is a must to understanding the functioning of Apache Nutch. So, have a look at the image below and you will get an overview of Apache Nutch plugin architecture.



I have also explained each component in detail, which will help you to understand them in depth:

- **Searcher**: Solr comes with a web interface. Solr allows you to run test searches. Access it at `http://localhost:8983/solr/admin/`. Enter some text into the **QueryString** box and click on **Search**. If your query matches any result, you should see an XML file containing the indexed pages of your websites.
- **Indexer**: This creates the inverted index from which the searcher extracts results. It uses Lucene storing indexes.

- **Web DB**: Web DB stores the document contents for indexing and later summarization by the searcher, along with information such as the link structure of the document space and the time each document was last fetched.
- **Fetcher**: Fetcher requests web pages, parses them, and extracts links from them. Nutch robot has been written entirely from scratch.

Summary

So that's the end of the first chapter. Let's discuss briefly what you have learned in this chapter. We started with the introduction of Apache Nutch. Then we moved on to the installation and verification of Apache Nutch and saw how crawling takes place with Apache Nutch. Then we had an introduction to Apache Solr and saw how Apache Solr can be integrated with Apache Nutch to index the data crawled by Apache Nutch. We covered crawling, crawling the web, and CrawlDB, as well as parsing and parse filters. And finally we saw how the Apache Nutch plugin can be created and also learned the Apache Nutch plugin architecture.

So that's all about the basics of Apache Nutch. Now let's see how deployment of Apache Nutch takes place and how we can apply sharding on Apache Solr in the next chapter.

2

Deployment, Sharding, and AJAX Solr with Apache Nutch

We have discussed the installation of Apache Nutch, crawling websites, and creating a plugin with Apache Nutch in the first chapter. This chapter covers deployment of Apache Solr on a particular server, such as Apache Tomcat, Jetty. Also, this chapter covers how sharding can take place with Apache Nutch using Apache Solr as a searcher.

In this chapter, we are going to cover following topics:

- Deployment of Apache Solr
- Sharding with Apache Solr
- Working with AJAX Solr

By the end of this chapter, you will be able to deploy Apache Solr on a server which contains the data crawled by Apache Nutch. You will also be able to perform sharding with data of Apache Nutch using Apache Solr. You will also be able to integrate AJAX with your running Apache Solr instance.

Deployment of Apache Solr

This part covers all the necessary steps which are required for performing deployment of Apache Solr.

Introduction of deployment

Installing, testing, and implementing a computer system or application is called deployment. This term can be used for any installation and testing, such as setting up a new network in an enterprise, installing a server farm, or implementing a new application on a distributed computing network. So after successful deployment of Apache Solr on Apache Tomcat, it's quite possible that by starting Apache Tomcat, Apache Solr will automatically start. No need to start it manually. It's a very important process because once Apache Nutch has crawled all the web pages for you and indexed them to Apache Solr, then the process needs to be started which will deploy Apache Solr on Apache Tomcat. After successful deployment of Apache Solr with Apache Tomcat, you can iterate on Apache Solr over a browser and can start searching on web pages which are crawled by Apache Nutch. You can search for any particular webpage. You can customize the search according to your needs. In short, you can make your own search engine by customizing Apache Solr. This is a very basic need for any web-based application because every application needs this type of scenario. You can integrate the deployed Apache Solr with your running application and get the benefit out of it.

Need of Apache Solr deployment

We are going to see how to deploy Apache Solr on Apache Tomcat. But the question is what is the need for that? I will give you an answer. Let's say you have crawled a number of websites using Apache Nutch. So, Apache Nutch has crawled a lot of pages for you. Now what? Here is when this deployment of Apache Solr comes into the picture. I will cover the sharding part in a later section. It's a kind of mechanism that divides the total number of URLs to different shards for bringing efficiency in Apache Solr searching. So, it's a very good concept that we are going to discuss in a further section. But for now, I can say that it's very much compulsory for Apache Solr to be deployed on any server as Apache Tomcat.

The prerequisites for deployment are as follows:

- JDK
- Tomcat
- Solr

For this deployment, I have used JDK 1.6, Apache Tomcat 7.0.41, and Solr 4.3.0.

Setting up Java Development Kit

The **Java Development Kit** (**JDK**) was developed by Oracle for Java developers. The JDK is an implementation of either Java SE, Java EE, or Java ME platforms. These platforms were released by Oracle Corporation in the form of binary products aimed at Java developers on Solaris, Linux, Mac OS X, or Windows. This is needed for running any Java-based application. So be sure you have JDK installed on your system. If not, please refer to `http://docs.oracle.com/javase/7/docs/webnotes/install/`. It will help you to set up the JDK and `JAVA_HOME` variable.

Setting up Tomcat

Apache Tomcat or Tomcat is an open source web server and Servlet container developed by **Apache Software Foundation** (**ASF**). It implements the Java Servlet and **JavaServer Pages** (**JSP**) specifications provided by Sun Microsystems and also provides a pure Java HTTP web server environment for Java code to run. It includes tools for configuration and management, but you can configure it by editing the XML configuration files. The following steps are given for the set up of Apache Tomcat:

1. Download the Apache Tomcat server from its official website at `http://tomcat.apache.org`.
2. Start Apache Tomcat by going to the `bin` directory of Apache Tomcat by executing the following commands:

   ```
   cd <Directory where your Apache Tomcat resides>/bin
   ./startup.sh
   ```

   If successful, you will get the following output:

```
root@abdulbasit:~/Softwares/apache-tomcat-7.0.41/bin# ./startup.sh
Using CATALINA_BASE:   /home/abdulbasit/Softwares/apache-tomcat-7.0.41
Using CATALINA_HOME:   /home/abdulbasit/Softwares/apache-tomcat-7.0.41
Using CATALINA_TMPDIR: /home/abdulbasit/Softwares/apache-tomcat-7.0.41/temp
Using JRE_HOME:        /usr/lib/jvm/jdk1.7.0
Using CLASSPATH:       /home/abdulbasit/Softwares/apache-tomcat-7.0.41/bin/b
root@abdulbasit:~/Softwares/apache-tomcat-7.0.41/bin#
```

3. Test it by opening `http://localhost:8080` on your browser, as Apache Tomcat is by default running on port 8080. If you have changed the port, then you need to put that port number instead of 8080.

4. You can stop Apache Tomcat by executing the following commands:

```
cd <Directory where your Apache Tomcat resides>/bin
./shutdown.sh
```

   If successful, you will get the following output:

```
root@abdulbasit:~/Softwares/apache-tomcat-7.0.41/bin# ./shutdown.sh
Using CATALINA_BASE:   /home/abdulbasit/Softwares/apache-tomcat-7.0.41
Using CATALINA_HOME:   /home/abdulbasit/Softwares/apache-tomcat-7.0.41
Using CATALINA_TMPDIR: /home/abdulbasit/Softwares/apache-tomcat-7.0.41/temp
Using JRE_HOME:        /usr/lib/jvm/jdk1.7.0
Using CLASSPATH:       /home/abdulbasit/Softwares/apache-tomcat-7.0.41/bin/b
root@abdulbasit:~/Softwares/apache-tomcat-7.0.41/bin#
```

   The other method for stopping Apache Tomcat is by typing the following command on the terminal where your Apache Tomcat is currently running:

```
Ctrl + c
```

Setting up Apache Solr

As I already covered the introduction of Apache Solr in *Chapter 1*, *Getting Started with Apache Nutch*, I will not cover that part here. I have used Apache Solr 4.3.0 for this deployment.

The following steps are given for setting up Apache Solr:

1. Download Apache Solr 4.3.0 from the Apache official website at `http://archive.apache.org/dist/lucene/solr/4.3.0/`.
2. Unzip Solr 4.3.0 to your respective directory. Let's say I have extracted it into `/usr/local`. So, the path of my Solr 4.3.0 is `/usr/local/solr-4.3.0`. So, follow the next steps accordingly.
3. Make a folder with the name `SOLR_HOME` in the respective folder of your choice. Let's say I have created the `SOLR_HOME` directory in `/usr/local`. So, the path of my `SOLR_HOME` directory would be `/usr/local/SOLR_HOME`.
4. Go to the `solr` directory which you will find in `/usr/local/solr-4.3.0/examples/solr`. Copy all the files which are inside it and paste them into `/usr/local/SOLR_HOME`. If you have started your Apache Solr already, then you will get a server-shutting-down exception. So, we have set up Apache Solr in the `SOLR_HOME` directory. We have also copied all the necessary files to this directory.

5. Go to the `solr` directory which you will find in `/usr/local/SOLR_HOME`. You will see two directories called `collection1` and `bin`. Copy these two directories and paste them in the `Solr_Home` directory. If Solr has already started and if `lib` directory is not copied, you will get an exception as the server error `filterstart`. The `collection1` and `bin` directories contain the necessary configuration files.
6. Copy JAR files called `log4j-1.2.16.jar`, `slf4j-api-1.6.6.jar`, and `slf4j-log4j12-1.6.6.jar`, which you will find in `/usr/local/solr-4.3.0/example/lib/ext` and also copy the `log4j.properties` file, which you will find in `/usr/local/solr-4.3.0/example/resources` and paste them in the `lib` directory which you will find in `<Respective directory where your Apache Tomcat resides>/lib`. You also need to download `commons-logging-1.1.3.jar` or any other respective version of `commons-logging jar` and paste it in the same `lib` directory. Otherwise, a logging error will appear. The `log4j-1.2.16.jar`, `slf4j-api-1.6.6.jar`, and `slf4j-log4j12-1.6.6.jar` files are used for logging mechanism. So when you run Apache Solr, if these JARs are not present, then you will not get any logs. The `log4j.properties` file contains the required properties for the logging configuration. The `commons-logging-1.1.3.jar` file is also used for logging mechanism.
7. Copy the `solr-4.3.0.war` file which you will find in `/usr/local/solr-4.3.0/dist` and paste it in the `webapps` directory which you will find in `<Respected directory where your Apache Tomcat resides>`. Make sure that you rename `solr-4.3.0.war` as `solr`. We are doing this for running Apache Solr in Tomcat. So, this WAR file will be deployed into Apache Tomcat when Apache Tomcat starts for the first time, and it will create the directory called `solr-4.3.0` inside the `webapps` directory, which is called an application of Apache Solr.
8. If Apache Tomcat has already started, then it will create a directory called `solr` in `<Respected directory where your Apache Tomcat resides>/webapps`. If Apache Tomcat has not started, then just start it as mentioned in the preceding *Setting up Tomcat* section.
9. Edit the `web.xml` file which you will find in `<Respected directory where your Apache Tomcat resides>/webapps/solr/WEB-INF` by adding the following content:

```
<env-entry>
  <env-entry-name>solr/home</env-entry-name>
  <env-entry-value>Path to your solr home</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

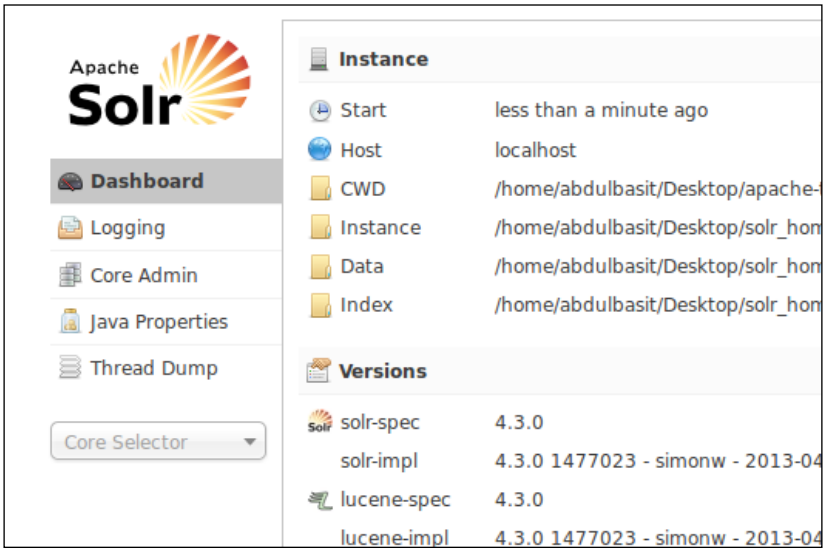
10. The above configuration will be commented by default. So uncomment it and give your Solr directory path which you will find in `/usr/local/Solr_Home` in `<env-entry-value>Path to your solr home</env-entry-value>`. So this configuration will tell Apache Tomcat where Apache Solr resides and it will use that path of Apache Solr whenever Apache Tomcat starts.
11. After modification, your preceding configuration will look as follows:

```
<env-entry>
  <env-entry-name>solr/home</env-entry-name>
  <env-entry-value>/usr/local/solr_home/solr</env-entry
    -value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Running Solr on Tomcat

We are done with all the configurations of Apache Solr for deploying it on Apache Tomcat. Now, it's time to actually run Apache Solr on Apache Tomcat and check whether it's functioning properly or not. You need to make sure that all the logs related to Apache Solr need to appear on the Apache Tomcat console when you start working on Apache Solr on the browser. If it's not appearing, then you need to check your configuration once again for what you have missed:

1. Finally, start your Apache Solr which is deployed on Apache Tomcat by entering `http//localhost:8080/solr` on your browser.
2. If successful, you will get an output like the following:



3. Check the logs on the Apache Tomcat console to see whether they are appearing or not. If yes, then you have configured it correctly. The following screenshot will be the output of Apache Tomcat logs:

4. If not, then you need to check your configuration once again to find out what you have missed.

So, that's all for the deployment of Apache Solr on Apache Tomcat. Now, let's move over to the next section.

Sharding using Apache Solr

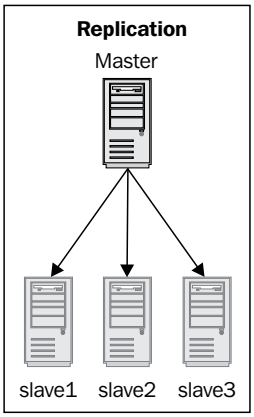
This section is going to describe all the necessary steps which are required to perform sharding using Apache Solr.

Introduction to sharding

When your data becomes too large for a single node, you can break it down into a number of sections by creating one or more shards. Each shard will be a portion of the logical index or core. A shard is the terminology which means splitting a core data over a number of servers or nodes. Take an example of representing each state; you might have a shard available for data which represents each state or it might be different categories which you want to make them search independently, but they are actually often combined. Load balancing comes into the picture whenever load increases in your application. Load means that the number of requests/hits is increased and your application is not able to handle that many requests. At this time, load balancing is required which will balance your application load and divide the load among machines, so that your application can run smoothly. Division of a load happens by transferring an incoming request to different machines. Whenever any request comes to the master machine, it will transfer that request to any slave machine. The master machine is not responsible for handling all requests. Load will decrease from the master machine and it will be divided among the slave machines. Scalability means whenever load increases, the number of servers increases. This is a simple definition of scalability. Scalability is a very basic requirement for any application. Your application must be scalable enough whenever load increases on it. So, scalability is required for handling load on your application. So, it's very important functionality of Apache Solr which is highly used in real-time applications. With this, handling load is almost a trivial task.

If users are limited in performing indexing on your application, then you can apply sharding. But if the number of users is increased, then sharding might not be a good option. In case many users are performing indexing simultaneously, then sharding isn't the answer to handling high traffic. You should use Solr replication. This will distribute complete copies of the master server to one or more slave servers. The job of the master server will be to update the index and all the queries will be handled by the slave servers. This division will make Apache Solr scalable and Apache Solr can provide quick responses against queries over large search volumes. Whenever load increases, more and more servers will be required to handle this load. This involves a master-slave configuration in which one server would be the master Solr server and the rest are the slave Solr servers. So, the master server will divide that load among the slave Solr severs whenever load increases on it. This increases Apache Solr's efficiency. You should use replication instead of sharding. In this case, we would like to set up with multiple Solr instances. They all need to have constant configuration. For configuring, we are going to use the `schema.xml` file which resides inside `$SOLR_HOME` which is a directory where your Apache Solr resides.

The following figure shows how load is divided from the master machine to the slave machines:



Replicating the entire index on multiple servers

A request will be first handled by the master server. The master server will decide to which slave server the request needs to be sent. Then, the master server will send the particular request to the particular slave server. The slave server will give response to that request. This is how Solr replication works. Update commands are sent to any server with distributed indexing organized properly. Whenever a new document needs to be added or deleted, then that request would be forwarded to that slave machine which has the support of a hash function of the distinctive document ID. The `commit` and `deleteByQuery` commands are sent to each server in shards.

Use of sharding with Apache Nutch

We are going to see how sharding takes place with Apache Nutch using Apache Solr. But before that, let's discuss why it is required. At this point, you must be aware of deployment of Apache Solr on Apache Tomcat. Now let's have a look at a practical scenario. Let's say you have crawled one million URLs and indexed them to Apache Solr. Now to make Apache Solr efficient in searching, you need to apply some mechanism on it. This is where sharding will be used. Sharding will divide the traffic by dividing the total number of URLs into different shards, which makes Apache Solr efficient in searching and, in addition, Apache Solr gives the higher throughput while searching. So in real-time applications, all these components will be integrated and this makes your application efficient in a web environment.

Distributing documents across shards

As explained previously, the total number of documents will be divided among a number of shards and each shard will be responsible for a certain amount of documents.

You can get all the documents indexed on every shard of your server. Solr doesn't embody out of the box support for distributed indexing. But our technique will be straightforward and you can send documents to any shard for indexing. A simple hashing system would additionally work. The formula would be as follows:

```
uniqueId.hashCode() % numServers
```

By using this approach, you will get an idea that a document needs to be updated or deleted.

Sharding Apache Solr indexes

Now, we'd like to pick the right quantity of shards. A group that is supposed to be designed is one among those variables that we must have before the final deployment. After making our collection, we can't modify the number of shards. We can only add a lot of replicas. Of course this comes with consequences—if we have chosen the number of shards incorrectly, we might find ourselves with a very low shards count. The only way to go would be making a brand new assortment with the correct number of shards and re-index our knowledge. With the release of Apache Solr 4.3, we are currently ready to split shards of our collections.

Single cluster

Now we tend to check the new shard-split functionality. We try and run a tiny low and single cluster containing a single Apache Solr instance with embedded ZooKeeper and use the instance assortment given with Apache Solr. The following steps are given for this setup:

1. Start your Apache Solr by going to the `example` directory which you will find in `<Respective directory where your Apache Solr resides>` and by executing following command:

```
java -Dbootstrap_confdir=./solr/collection1/conf -
  Dcollection.configName=collection1 -DzkRun -DnumShards=1 -
  DmaxShardsPerNode=2 -DreplicationFactor=1 -jar start.jar
```

If successful, you will get an output like the following:

An explanation of the preceding command is as follows:

- For defining the name of the collection, the following argument will be provided:

  ```
  Dcollection.configName=collection1
  ```

- For starting embedded ZooKeeper, the following argument will be provided:

  ```
  -DzkRun
  ```

- For defining the number of shards, the following argument will be provided:

  ```
  -DnumShards=1
  ```

- You can increase the number of shards by providing an incremented value in the following argument:

  ```
  -DnumShards
  ```

- For defining the maximum number of shards per node, the following argument will be provided:

  ```
  -DmaxShardsPerNode=2
  ```

- For defining the replication factor, the following argument will be used:

  ```
  DreplicationFactor=1
  ```

- For starting Apache Solr, the following argument will be provided:

```
-jar start.jar
```

In the preceding command, `-jar` is an argument for defining JAR and `start.jar` is the actual JAR which will start Apache Solr.

2. Go to the `exampledocs` directory which you will find in `<Respective directory where your Apache Solr resides>/example` and execute the following command:

```
java -jar post.jar *.xml
```

If successful, you will get an output as follows:

```
root@abdulbasit:~/Desktop/solr-4.3.0/example/exampledocs# java -jar
SimplePostTool version 1.5
Posting files to base url http://localhost:8983/solr/update using c
POSTing file gb18030-example.xml
POSTing file hd.xml
POSTing file ipod_other.xml
POSTing file ipod_video.xml
POSTing file manufacturers.xml
POSTing file mem.xml
POSTing file money.xml
POSTing file monitor2.xml
POSTing file monitor.xml
POSTing file mp500.xml
POSTing file sd500.xml
POSTing file solr.xml
POSTing file utf8-example.xml
POSTing file vidcard.xml
14 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/update.
Time spent: 0:00:01.477
```

The preceding command will index all XML files which reside in the `exampledocs` directory of Apache Solr. Apache Solr needs these XML files to perform sharding on them.

3. The number of indexed documents can be checked with the following command:

```
curl 'http://localhost:8983/solr/
  collection1/select?q=*:*&rows=0'
```

The response returned by Solr is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader"><int name="status">0</int><int
  name="QTime">5</int><lst name="params"><str
  name="q">*:*</str><str
  name="rows">0</str></lst></lst><result name="response"
  numFound="32" start="0"></result>
</response>
```

As you can see, we have got 32 documents in our collection.

Splitting shards with Apache Nutch

In this section, we will divide a whole shard into a number of shards. So, the total number of documents will be divided into this number of shards. And after that instead of having one collection, we will have a different number of shards. The following command is given to divide the shards:

```
curl 'http://localhost:8983/solr/admin/collections?
  action=SPLITSHARD&collection=collection1&shard=shard1'
```

If successful, we will get an output as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader">
...
</lst>
</response>
```

Cleaning up with Apache Nutch

First of all in order to see the data in new shards, we need to run the `commit` command against our collection. This can be done by using the following command:

```
curl 'http://localhost:8983/solr/collection1/update'
  --data-binary '<commit/>' -H 'Content-type:application/xml'
```

If successful, you will get an output as follows:

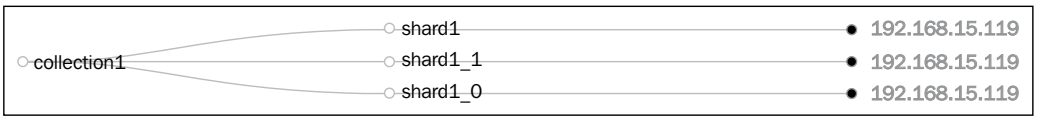
```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader"><int name="status">0</int><int
  name="QTime">173</int></lst>
</response>
```

Splitting cluster shards

After splitting a shard cluster, we will get an output on Apache Solr on the browser as follows:

Cluster shard before splitting

The preceding screenshot shows the single shard cluster before splitting. You can see that traffic is passing from only one shard. You will see in the following screenshot, which shows the multiple shard clusters, in which traffic is going from multiple shards.



Cluster shard after splitting

As you see in the first diagram on this page, we have one collection point before cluster shard splitting. After cluster shard splitting, it's divided into three completely different shards: **shard1**, **shard1_0**, and **shard_1**, as you can see in the preceding screenshot. Every shard will contain a portion of documents from the original **shard1**. Some documents would go to **shard1_1** and some of them should be placed in **shard1_0** and the total number of shards will be displayed in the main shard, that is **shard1**. So this is how the division of documents is done in Apache Solr sharding.

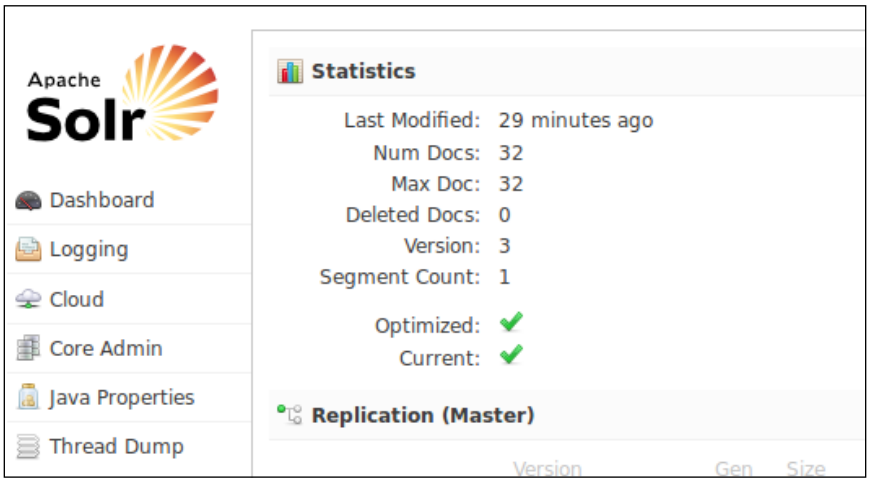
Checking statistics of sharding with Apache Nutch

Now, it's time to see the output on Apache Solr on the browser. It will show you the statistics of every shard such as how many documents each shard contains and the details related to that shard will be displayed.

For checking statistics of the main shard, that is **shard1**, use the following URL:

```
http://localhost:8983/solr/#/collection1
```

If successful, you will get an output as follows:

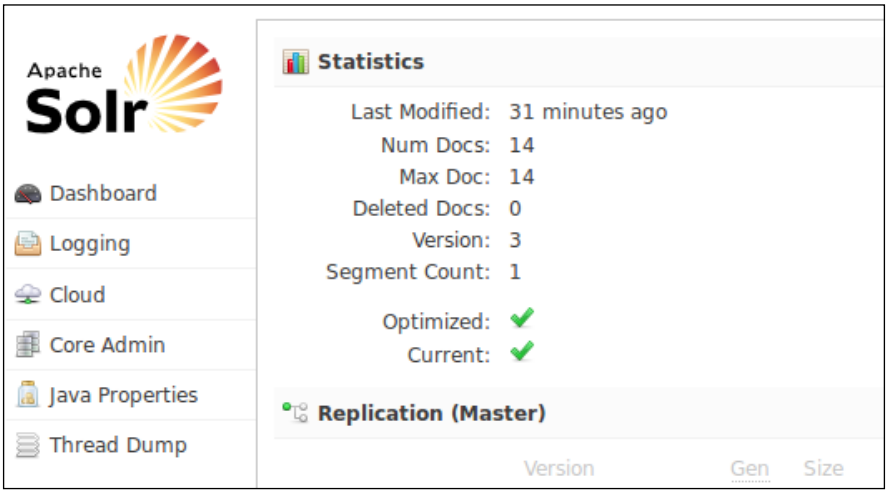


You can see that the total number of documents is 32 represented by **Num Docs: 32** in the **Statistics** tab.

For checking statistics of **shard1_0**, use the following URL:

```
http://localhost:8983/solr/#/collection1_shard1_0_replica1
```

If successful, you will get an output as follows:

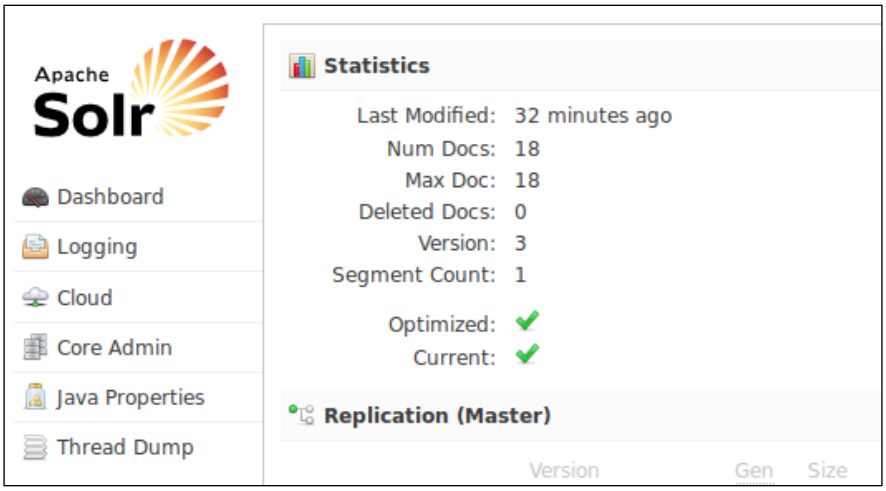


You can see that the total number of documents is 14 represented by **Num Docs: 14** in the **Statistics** tab.

For checking statistics of **shard1_1**, use the following URL:

```
http://localhost:8983/solr/#/collection1_shard1_1_replica1
```

If successful, you will get an output as follows:



You can see that the total number of documents is 18 represented by **Num Docs: 18** in the **Statistics** tab.

So, 32 documents are divided into two shards. **shard1_0** contains 14 documents and **shard1_1** contains 18 documents. This division of documents is handled by Apache Solr internally. So, we might not be able to predict which shard contains how many documents.

The final test with Apache Nutch

This is the final step for sharding with Apache Solr. You have to check whether your documents are available in the shards created by the `SPLITSHARD` action. Use the following URL to check the availability of documents in shards:

```
http://localhost:8983/solr/collection1/select?q=*:*&rows=100&fl=id,[shard]&indent=true
```

If successful, you will get an output as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">7</int>
  <lst name="params">
    <str name="fl">id,[shard]</str>
    <str name="q">*:*</str>
    <str name="rows">100</str>
  </lst>
</lst>
<result name="response" numFound="32" start="0" maxScore="1.0">
  <doc>
    <str name="id">GB18030TEST</str>
    <str name="[shard]">192.168.56.1:8983
      /solr/collection1_shard1_0_replica1/</str></doc>
...
...
...
  <doc>
    <str name="id">100-435805</str>
    <str name="[shard]">192.168.56.1:8983
      /solr/collection1_shard1_1_replica1/</str></doc>
</result>
</response>
```

As you can see, documents are coming from both shards, that is **shard1_1** and **shard1_0** ,which is again what we expected. So just check the output, whether the documents which you indexed to Apache Solr are appearing or not.

So that's the end of the sharding with Apache Nutch using Apache Solr. It's a very useful technique when you work in real-time applications. So the basic agenda behind this implementation is that you should use this technique in your real-time applications where you have crawled millions of web pages using Apache Nutch and indexed them to Apache Solr. Apply sharding on them and get the best result from Apache Solr.

Working with AJAX Solr

AJAX Solr is a JavaScript library for making user interfaces for Apache Solr. It is JavaScript framework-agnostic, however, it needs an AJAX implementation to communicate with Apache Solr. As such, you'll use the library whether or not you develop jQuery, MooTools, Prototype, Dojo, or any other framework. You will have to define only a manager object that extends the provided abstract manager object, and define the function `executeRequest()` in the object. A jQuery-compatible manager is provided at `managers/Manager.jquery.js`.

AJAX Solr began as a fork of SolrJS created by *Matthias Epheser*. We need to understand it very clearly to integrate this with our running Apache Solr.

Architectural overview of AJAX Solr

AJAX Solr loosely follows the **Model-View-Controller** (**MVC**) pattern. The **Parameter Store** is the model, that stores the Solr parameters and thus, the state of the appliance. The **Manager** is the controller; it connects to the Parameter Store and asks for the required details, sends requests to Solr, and delegates the response to the widgets for rendering. The widgets are used to view data.

Applying AJAX Solr on Reuters' data

Reuters is actually a dataset on which we are going to apply the AJAX Solr mechanism. You can download the Reuters' data from `http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`. Before we begin, we tend to write the **Hypertext Mark-up Language** (**HTML**) to the JavaScript widgets that can be integrated with each other. In order to proceed further, this HTML can typically be the non-JavaScript version of your search interface that we would like to improve with unobtrusive JS. Here we use jQuery and jQuery UI.

Running AJAX Solr

If you want to integrate AJAX with your Apache Solr, you need to do a certain configuration. It's a very good feature provided by this library. It will make the user UI-rich and user-friendly. So, it's a very useful technique in real-time applications that will load your data on Apache Solr without refreshing the whole web page.

The following steps are given for integrating AJAX Solr with your running local instance of Apache Solr:

1. Download this tarball `reuters_data.tar.gz` which can be downloaded from `https://github.com/downloads/evolvingweb/ajax-solr/reuters_data.tar.gz`, which contains a Solr index of the Reuters information. Replace the `data` directory of Apache Solr which you will find in `<Respected directory where your Apache Solr resides>/example/solr/collection1` of your Solr instance with this `data` directory of tarball which you will find in `<Respected directory where your reuters_data resides>`.
2. Put the following configuration in the `schema.xml` file which you will find in `<Respected directory where your Apache Solr resides>/example/solr/collection1/conf`:

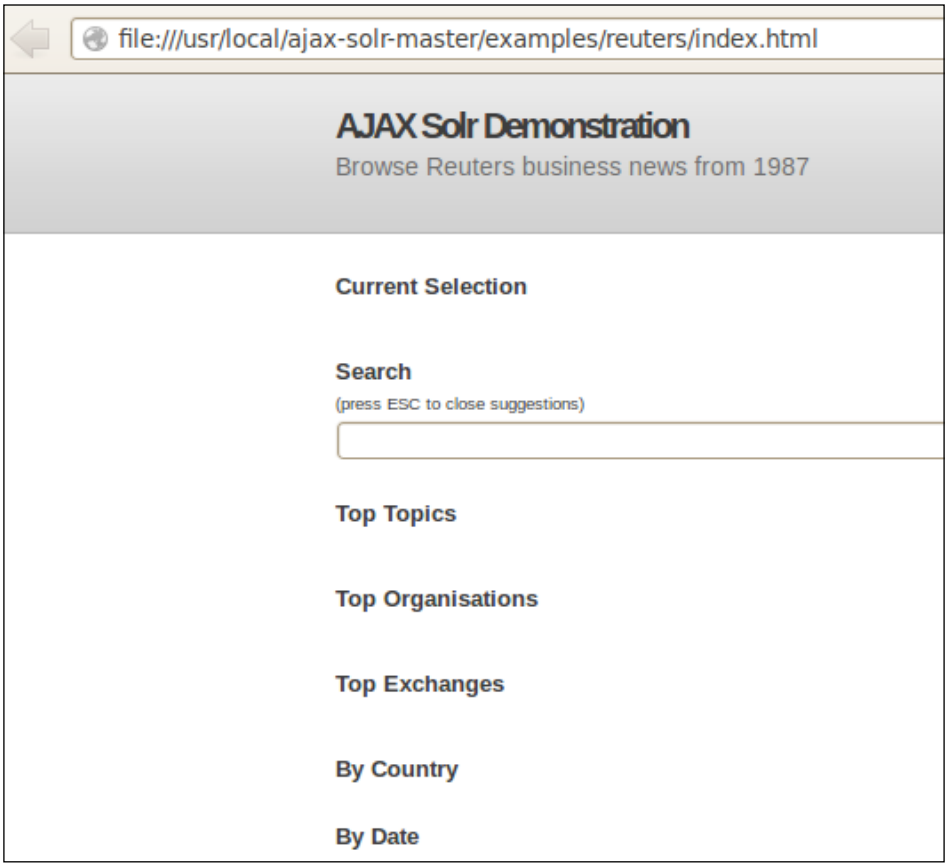
```
<field name="id" type="string" indexed="true" stored="true"
  required="true" multiValued="false" />
<field name="sku" type="text_en_splitting_tight"
  indexed="true" stored="true" omitNorms="true"/>
<field name="name" type="text_general" indexed="true"
  stored="true"/>
<field name="manu" type="text_general" indexed="true"
  stored="true" omitNorms="true"/>
<field name="cat" type="string" indexed="true"
  stored="true" multiValued="true"/>
<field name="features" type="text_general" indexed="true"
  stored="true" multiValued="true"/>
<copyField source="title" dest="text"/>
<copyField source="author" dest="text"/>
<copyField source="description" dest="text"/>
<copyField source="keywords" dest="text"/>
<copyField source="content" dest="text"/>
<copyField source="content_type" dest="text"/>
<copyField source="resourcename" dest="text"/>
<copyField source="url" dest="text"/>
<field name="date" type="pdate" indexed="true"
  stored="true" multiValued="true" omitNorms="true"
  termVectors="true" />
<field name="dateline" type="string" indexed="true"
  stored="true" multiValued="true" omitNorms="true"
  termVectors="true" />
<copyField source="dateline" dest="allText"/>
```

3. Modify `reuters.js` which you will find in `<Respected directory where your ajax-solr-master directory resides>/examples/reuters/js` by putting the following configuration into it:
    - Find the `solrUrl` field in `reuters.js` and set your running Apache Solr URL there which will be `http://localhost:8983/solr` by default if you haven't changed anything.
    - Find `facet.field` in `reuters.js` and update it with the fields you want to facet. For example, `title`.
    - Remove `f.topics.facet.limit` and `f.countrycodes.facet.limit` from `reuters.js` unless you have used these fields in your Apache Solr.
    - Remove all `facet.date` parameters unless you have used the date field to facet in your Apache Solr.

    You can update or remove tag cloud, autocomplete, country code, and calendar widgets from `reuters.js`. You can set the associated Solr fields for the tag cloud by changing the value of `var` fields in `reuters.js`. For example, `title`, `url`, `content`.

4. Apache Nutch uses a content field instead of a text field as Reuters. Modify `ResultWidget.js` which you will find in `examples/reuters/widgets`. Replace all occurrences of `doc.text` with `doc.content` in the `template` method. Apache Nutch is not using any dateline field. So remove all occurrences of `doc.dateline + ' ' +`.

5. Just open the `index.html` file in your browser which you will find in `examples/reuters/index.html`. If successful, you will get an output as follows:



Now you can enjoy playing with AJAX on Apache Solr. So this ends the integration of AJAX with your running instance of Apache Solr.

Summary

So, that's the end of this chapter. Let's quickly revise what we have learned. We started with deployment of Apache Solr on Apache Tomcat. In this, we started with an introduction on deployment and then we moved over to the installation steps and got an idea of JDK, Apache Tomcat, and what the prerequisites for Apache Solr deployment are. Then, we covered sharding on Apache Solr. In this, we started with an introduction on sharding and then we covered how sharding takes place on Apache Solr. And finally, we saw a very key concept called AJAX Solr which can be highly used in real-time applications. In this, we covered AJAX Solr, architectural overview of AJAX Solr, AJAX Solr Reuters, and finally running AJAX Solr.

I hope you enjoyed reading this chapter. In the next chapter, we will discuss how Apache Nutch can be integrated with Apache Hadoop and how Apache Nutch can be integrated with Eclipse in the same way.

3
Integration of Apache Nutch with Apache Hadoop and Eclipse

We have discussed in *Chapter 2*, *Deployment, Sharding, and AJAX Solr with Apache Nutch*, how deployment takes place with Apache Solr and how we can apply Sharding using Apache Solr. We have also covered integrating AJAX with our running Apache Solr instance. In this chapter, we will see how we can integrate Apache Nutch with **Apache Hadoop**, and we will also see how we can integrate Apache Nutch with **Eclipse**. Apache Hadoop is a framework which is used for running our applications in a cluster environment. Eclipse will be used as an **Integrated Development Environment** (**IDE**) for performing crawling operations with Apache Nutch. We will discuss in detail about this in the coming sections. So, we will first start with the integration of Apache Nutch with Apache Hadoop. And then, we will move towards the integration of Apache Nutch with Eclipse. So let's get started.

In this chapter, we are going to cover the following topics:

- Integrating Apache Nutch with Apache Hadoop
    - Introducing Apache Hadoop
    - Introducing Apache Nutch integration with Apache Hadoop
    - Installing Apache Hadoop and Apache Nutch
    - Setting up the deployment architecture of Apache Nutch

- Configuring Apache Nutch with Eclipse
    - Introducing the Apache Nutch configuration with Eclipse
    - Installing and building Apache Nutch with Eclipse
    - Crawling in Eclipse

By the end of this chapter, you will be able to set up Apache Nutch on Apache Hadoop in your own environment. You will also be able to perform crawling in Apache Nutch by using Eclipse.

Integrating Apache Nutch with Apache Hadoop

In this section, we will see how Apache Nutch can be integrated with Apache Hadoop. So, we will start by introducing Apache Hadoop. Then, we will have some basic introduction of integration of Apache Nutch with Apache Hadoop. Lastly, we will move over to the configuration part of Apache Hadoop and Apache Nutch. We will also see how we can deploy Apache Nutch on multiple machines.

Introducing Apache Hadoop

Apache Hadoop is designed for running your application on servers where there will be a lot of computers, one of them will be the master computer and the rest will be the slave computers. So, it's a huge data warehouse. Master computers are the computers that will direct slave computers for data processing. So, processing is done by slave computers. This is the reason why Apache Hadoop is used for processing huge amounts of data. The process is divided into the a number of slave computers, which is why Apache Hadoop gives the highest throughput for any processing. So, as data increases, you will need to increase the number of slave computers. That's how the Apache Hadoop functionality runs.

Apache Nutch can be easily integrated with Apache Hadoop, and we can make our process much faster than running Apache Nutch on a single machine. After integrating Apache Nutch with Apache Hadoop, we can perform crawling on the Apache Hadoop cluster environment. So, the process will be much faster and we will get the highest amount of throughput.

Installing Apache Hadoop and Apache Nutch

In this section, we will see how we can configure Apache Hadoop and Apache Nutch in our own environment. After installing these, you can perform the crawling operation in Apache Nutch, which will be run on the Apache Hadoop cluster environment.

Downloading Apache Hadoop and Apache Nutch

Both Apache Nutch and Apache Hadoop can be downloaded from the Apache websites. You need to check out the newest version of Nutch from the source after downloading. As an alternative, you can pick up a stable release from the Apache Nutch website.

Setting up Apache Hadoop with the cluster

Apache Hadoop with Cluster setup does not require a huge hardware to be purchased to run Apache Nutch and Apache Hadoop. It is designed in such a way that it makes the maximum use of hardware. So, we are going to use six computers to set up Apache Nutch with Apache Hadoop. My computer configuration in the cluster will have Ubuntu 10.04 installed. The names of our computers are as follows:

- `reeshucluster01`
- `reeshucluster02`
- `reeshucluster03`
- `reeshucluster04`
- `reeshucluster05`
- `reeshucluster06`

To start our master node we use the `reeshucluster01` computer. By master node, I mean that this will run the Hadoop services, which will coordinate with the slave nodes (the rest of the computers). In this case, all the remaining nodes will be the slave nodes, that is, `reeshucluster02`, `reeshucluster03`, and so on. And it's the machine on which we will perform our crawl. We have to set up Apache Hadoop on each of the above clusters. The steps for setting up Apache Hadoop on these clusters are described in the following sections.

Installing Java

Java is a programming language by Sun Microsystems. It is fast, secure, and reliable. Java is used everywhere—in laptops, data centers, game consoles, scientific supercomputers, cell phones, Internet, and so on. My cluster configuration consists of Java 6, that's why I have explained just Java 6. If your cluster configuration has Java 7 installed, you need to change it accordingly. You can refer to `http://askubuntu.com/questions/56104/how-can-i-install-sun-oracles-proprietary-java-6-7-jre-or-jdk` for installing Java 7. The steps for installing Java 6 are given as follows. You can always refer to `http://www.oracle.com/technetwork/java/javase/install-linux-self-extracting-138783.html` if you are facing any difficulty in installing:

1. Download the 32-bit or 64-bit Linux **compressed binary file** from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. It has a `.bin` file extension.
2. Give it permissions to execute and extract it using the following command:

   ```
   #chmod a+x [version]-linux-i586.bin
   ./[version]-linux-i586.bin
   ```

3. During installation, it will ask you to register—press *Enter*. Firefox will open with the registration page. Registration is optional. JDK 6 package is extracted into the `./jdk1.6.0_x` directory, for example, `./jdk1.6.0_30`.
4. Let's rename it as:

   ```
   #mv jdk1.6.0_30 java-6-oracle
   ```

5. Now, move the `JDK 6` directory to`/usr/lib`.

   ```
   #sudo mkdir /usr/lib/jvm
   #sudo mv java-6-oracle /usr/lib/jvm
   ```

6. Run the self-extracting binary.
7. Execute the downloaded file, which is prepended by the path to it. For example, if the file is in the current directory, prepend it with `./` (necessary if `.` is not in the `PATH` environment variable):

   ```
   # ./jdk-6u <version>-linux-i586.bin
   ```

8. The binary code license is displayed, and you are prompted to agree to its terms.
9. The JDK files are installed in a directory called `jdk1.6.0_<version>` in the current directory.
10. Delete the `bin` file if you want to save disk space.

11. Finally, test it by using the following command:

```
#java -version
#javac -version
```

12. The preceding commands should display the Oracle version installed. For example, **1.6.0_30**.
13. If all succeeds, you will get following output:

```
C:\Users\USER>java -version
java version "1.6.0_16"
Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
Java HotSpot(TM) 64-Bit Server VM (build 14.2-b01, mixed mode)
```

The preceding screenshot will show you that your JDK is successfully installed and your Java is running correctly.

Downloading Apache Hadoop

I have used Apache Hadoop 1.1.2 for this configuration. I have tried to cover everything correctly. Still, you can always refer to `http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/` for running Apache Hadoop on Ubuntu Linux for single-node clusters. Refer to `http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/` for running Apache Hadoop on Ubuntu Linux for multi-node cluster.

The steps for downloading Apache Hadoop are given as follows:

1. Download Apache Hadoop 1.1.2 branch distribution from `http://www.apache.org/dyn/closer.cgi/hadoop/common/`. Unzip Apache Hadoop using the relevant commands. If you're using Windows, you will have to use an archive program, such as WinZip or WinRar, for extracting. Fire the following command by going to the directory where your downloaded Apache Hadoop resides:

   ```
   $tar hadoop-1.1.2.tar.gz
   ```

   I have configured Apache Hadoop by taking a separate new user on my system as it's a good practice to do it. I have created a new user and given permission to Apache Hadoop to allow only this user to access Apache Hadoop. It's for security purposes (so that no other user can access Apache Hadoop). Not even the root user is able to access Apache Hadoop. You have to log in as this user to access Apache Hadoop.

2. The following command will add a new group called `hadoop`:

   ```
   $ sudo addgroup hadoop
   ```

3. The following command will add new user called `hduser`, and then adds this user to the `hadoop` group:

   ```
   $ sudo adduser --ingroup hadoop hduser
   ```

Configuring SSH

Apache Hadoop needs SSH, which stands for Secure Shell, to manage its nodes, that is, remote machines and your local machine. It is used to log into the remote system or the local system, and also performs necessary operations on a particular machine. So, you need to configure this in your local environment as well as in your remote environment for running Apache Hadoop. The commands and steps for configuring SSH are given as follows:

1. The following command will be used for logging into the `hduser`:

   ```
   user@ubuntu:~$ su - hduser
   ```

   It will ask for a password, if one is set. Enter the password and you will be logged into that user.

2. The following command will be used for generating a key. This key will be used to provide authentication at the time of login. Make sure you are logged in as an `hduser` before firing the following command.

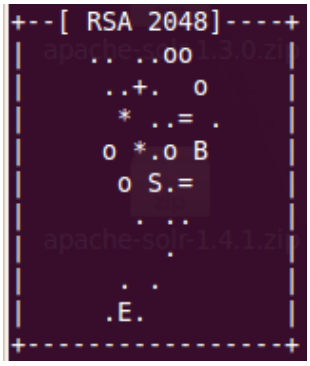
   ```
   hduser@ubuntu:~$ ssh-keygen -t rsa -P ""
   ```

3. You will get a result as follows:

   ```
   Generating public/private rsa key try.
   Enter move into that to save lots of the key
   (/home/hduser/.ssh/id_rsa):
   ```

4. Press *Enter* and you will get a result as follows:

   ```
   Created directory '/home/hduser/.ssh'.
   Your identification has been saved in /home/hduser/.ssh/id_rsa.
   Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
   The key fingerprint is:
   9b:82:ea:58:b4:e0:35:d7:ff:19:66:a6:ef:ae:0e:d2 hduser@ubuntu
   The key's random art image is:
   ```



The preceding screenshot is showing the generated key image.

The second line will make an RSA key pair with an empty password. Generally, using a blank password isn't recommended.

5. The following command will copy the generated key to the `authorized_keys` directory that you will find in `$HOME/.ssh`, where `$HOME` will be your home directory (that is, `/home/reeshu`). It's required for authentication at the time of SSH login.

```
$hduser@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >>
$HOME/.ssh/authorized_keys
```

6. The final step is to test the SSH setup by connecting to your local machine as the `hduser`. The following command will be used for testing:

```
hduser@ubuntu:~$ ssh localhost
```

7. You will get an output as follows:

```
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is d7:87:25:47:ae:02:00:eb:1d:75:4f:bb:44
:f9:36:26.
Are you sure you want to continue connecting (yes/no)?
Just type yes and press enter. You will get as follows:
Warning: Permanently added 'localhost' (RSA) to the list of known
hosts.
Linux ubuntu 2.6.32-22-generic #33-Ubuntu SMP Wed Apr 28 13:27:30
UTC 2010 i686 GNU/Linux
Ubuntu 10.04 LTS
hduser@ubuntu:~$
```

Disabling IPv6

Internet Protocol version 6 (**IPv6**) is the latest revision of **Internet Protocol** (**IP**). It is also the communication protocol which provides the location and an identification for routers and computers on networks. Sometimes this address creates problems in configuring Apache Hadoop. So, it is better to disable it. The configuration for disabling IPv6 is discussed next.

Sometimes, using `0.0.0.0` for various networking-related Apache Hadoop configuration results in Apache Hadoop binding to the address of IPv6 of the Ubuntu box. IPv6 may not be required when you are not connected to the IPv6 network. So at that time, you can disable it.

For disabling IPv6, open `sysctl.conf` (which you will find in `/etc`) and put the following configuration at the end of the file:

```
# disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Reboot your machine to apply the effect. To check whether IPv6 is enabled or not on your machine, the following command will be used:

```
$ cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

If it returns `0`, IPv6 is enabled; and if it returns `1`, IPv6 is disabled. And that's what we want. Another way of disabling IPv6 is by changing `hadoop-env.sh`, which you will find in `<Respected directory where your Apache Hadoop resides>/conf`. Open `hadoop-env.sh` and put the following line into it:

```
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

Installing Apache Hadoop

To install Apache Hadoop, download Apache Hadoop as discussed, and extract it to your preferred directory. I have used the `/usr/local` directory. So, I will follow this directory for the installation. You have to follow your directory. The following commands will be used to change the owner of all the files to the `hduser` user and the `hadoop` group cluster. So, we will assign permission to Apache Hadoop such that only `hduser` is able to access Apache Hadoop and not any other user:

- The following command will take you to the directory where Apache Hadoop resides:

  ```
  $ cd /usr/local
  ```

- The following command will extract Apache Hadoop:

  ```
  $ sudo tar xzf hadoop-1.0.3.tar.gz
  ```

- The following command will rename Apache Hadoop to `hadoop`:

  ```
  $ sudo mv hadoop-1.0.3 hadoop
  ```

- The following command will give `hduser` permission to access Apache Hadoop:

  ```
  $ sudo chown -R hduser:hadoop hadoop
  ```

- Open the `bashrc` file by going to the root directory, and type the following command:

  ```
  gedit ~/.bashrc
  ```

I have created `HADOOP_HOME` and `JAVA_HOME` where my Apache Hadoop and Java reside. You need to also create a location where your Apache Hadoop and Java will reside. It will set the Apache Hadoop and Java to our classpath, which is required for this configuration. Put the following configuration at the end of the file:

```
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-6-sun
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -l
export PATH=$PATH:$HADOOP_HOME/bin:$JAVA_HOME/bin
```

Required ownerships and permissions

In this section, we are going to cover configuration of data files, the network ports of Apache Hadoop on which Apache Hadoop listens, and so on. This setup will use **Hadoop Distributed File System** (**HDFS**), though there is only a single machine in our cluster. We need to create three directories. We will create directories called `app`, `hadoop`, and `tmp`. So, this will be like `/app/hadoop/tmp`. Apache Hadoop will use the `tmp` directory for its operations. Hadoop default configurations use `hadoop.tmp.dir`. It is a property that you will find in `core-site.xml`, which resides in `/usr/local/hadoop/conf` for local file system and HDFS. Therefore, you should not get surprised if, at some later point, you see Hadoop making the required directory mechanically on HDFS. The following command is used for creating this directory:

```
$ sudo mkdir -p /app/hadoop/tmp
```

The following command will allow the `/app/hadoop/tmp` directory to be accessed only by `hduser`:

```
$ sudo chown hduser:hadoop /app/hadoop/tmp
```

And if you want to tighten up security, change the value of `chmod` from 755 to 750, using the following command:

```
$ sudo chmod 750 /app/hadoop/tmp
```

The configuration required for Hadoop_HOME/conf/*

In this section, we will modify `core-site.xml`, `hdfs-site.xml` and `mapred-site.xml`, which reside inside the `conf` directory that you will find in `/usr/local/hadoop`. This configuration is required for Apache Hadoop to perform its operations:

- `core-site.xml`: The `fs.default.name` property is used by Apache Nutch to check the filesystem which it's attempting to use. Since we are working with the Apache Hadoop filesystem, we've pointed this to the `hadoop` master or name node. In this case it's `hdfs://reeeshu:54311t`, which is the location of our master name node. You must put your master node accordingly. This will provide information about our master node to Apache Hadoop. Put the following configuration into it:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://reeeshu:54311</value>
    <description>
       Where to find the Hadoop Filesystem through the network.
       Note 54311 is not the default port.
       (This is slightly changed from previous versions which
didnt have "hdfs")
    </description>
  </property>
</configuration>
```

- `hdfs-site.xml`: This file will be used to tell Apache Hadoop how many replications Apache Hadoop should use. The `dfs.replication` property will tell Apache Hadoop about the number of replications to be used. By default, the value is `1`. It means that there is no replication, that is, it is using only a single machine in a cluster. It should usually be three or more—in fact you should have a minimum in that range of operating nodes in your Hadoop cluster. The `dfs.name.dir` property will be used as the Apache Nutch name directory. The `dfs.data.dir` data directory is used by Apache Nutch for its operations. You must create these directories manually and give the proper path of those directories. Put the following configuration into it:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>

<property>
  <name>dfs.name.dir</name>
  <value>/nutch/filesystem/name</value>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>/nutch/filesystem/data</value>
</property>

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>

</configuration>
```

  The `dfs.name.dir` property is the directory used by the name node for storing, following, and coordinating the data for the info nodes. The `dfs.data.dir` property is the directory used by the data nodes for storing the particular filesystem's data blocks. This should often be expected to be similar on each node.

- `mapred-site.xml`: The distributed file system has name nodes and information nodes. When a client wants to govern a file in the filesystem, it will contact the name node. The name node will indicate which data node to contact for accepting the file. The name node is the organizer and will store what blocks area unit on what computers, and what must be replicated to completely different data nodes. The data node's area units are simply the workhorses. They are storing the particular files, serving them up on request, and so on. If you're running a name node and a data node on the same PC, it will still act over sockets as if the information node was on a different PC. Put the following configuration into it:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>reeeshu:54310</value>
  <description>
    The host and port that the MapReduce job tracker runs at. If
    "local", then jobs are run in-process as a single map and
    reduce task.
    Note 54310 is not the default port.
  </description>
</property>

<property>
  <name>mapred.map.tasks</name>
  <value>2</value>
  <description>
    define mapred.map tasks to be number of slave hosts
  </description>
</property>

<property>
  <name>mapred.reduce.tasks</name>
  <value>2</value>
  <description>
    define mapred.reduce tasks to be number of slave hosts
  </description>
</property>
```

```
<property>
  <name>mapred.system.dir</name>
  <value>/nutch/filesystem/mapreduce/system</value>
  <description>
    Define system directory. You have to manually create the
directory and give the proper path to the value property.
  </description>
</property>

<property>
  <name>mapred.local.dir</name>
  <value>/nutch/filesystem/mapreduce/local</value>
  <description>
    Define local directory. You have to manually create the
directory and give the proper path to the value property.
  </description>
</property>

</configuration>
```

The `mapred.system.dir` property stores the name of the directory that the mapreduce tracker uses to store its data. This is often only on the tracker and not on the mapreduce hosts.

Formatting the HDFS filesystem using the NameNode

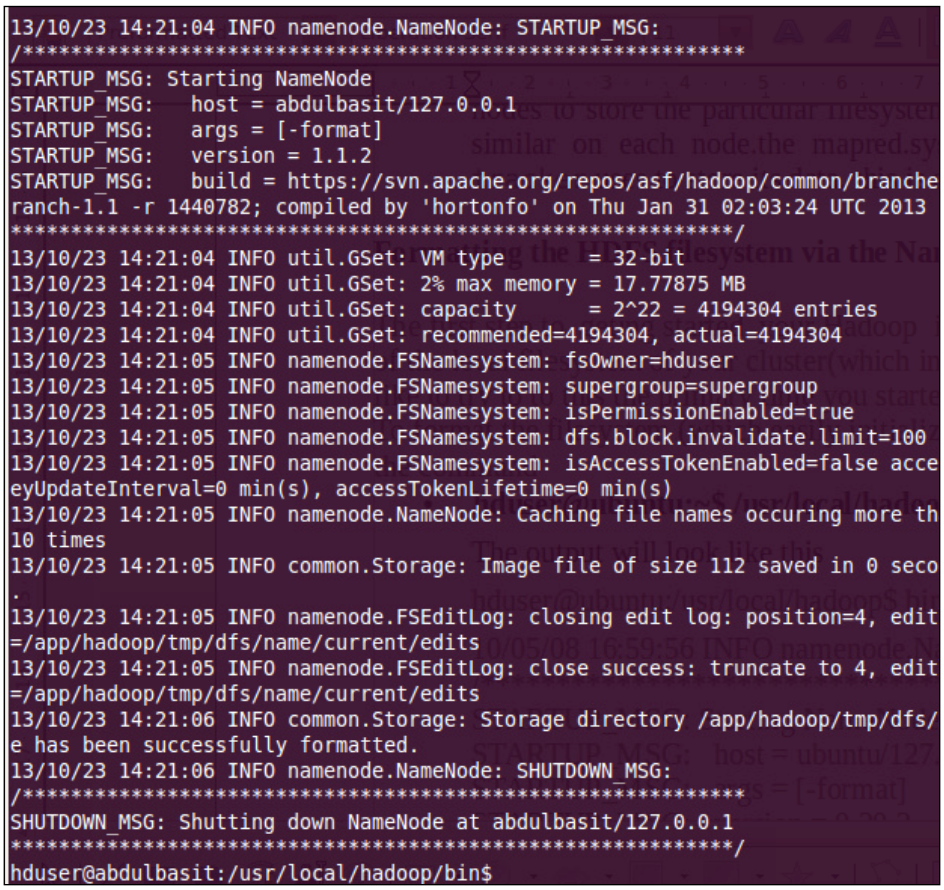
HDFS is a directory used by Apache Hadoop for storage purposes. So, it's the directory that stores all the data related to Apache Hadoop. It has two components: `NameNode` and `DataNode`. `NameNode` manages the filesystem's metadata and `DataNode` actually stores the data. It's highly configurable and well-suited for many installations. Whenever there are very large clusters, this is when the configuration needs to be tuned.

The first step for getting your Apache Hadoop started is formatting the Hadoop filesystem, which is implemented on top of the local filesystem of your cluster (which will include only your local machine if you have followed). The HDFS directory will be the directory that you specified in `hdfs-site.xml` with the property `dfs.data.dir` explained previously.

To format the filesystem, go to the respective directory where your Apache Hadoop resides by terminal. In my case, it is in `/usr/local`. Make sure that you are logged in as a `hduser` before hitting the following command:.

```
hduser@ubuntu:~$ /usr/local/hadoop/bin/hadoop namenode -format
```

If all succeeds, you will get an output as follows:



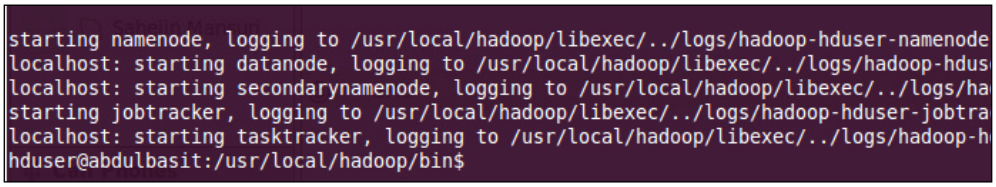
The preceding screenshot shows that your HDFS directory is formatted successfully.

Starting your single-node cluster

Now, we are done with the setup of the single-node cluster of Apache Hadoop. It's time to start Apache Hadoop and check whether it is running up to the mark or not. So, run the following command for starting your single-node cluster. Make sure you are logged in as `hduser` before hitting the following command:.

```
hduser@ubuntu:~$ /usr/local/hadoop/bin/start-all.sh
```

If all succeeds, you will get the output as follows:

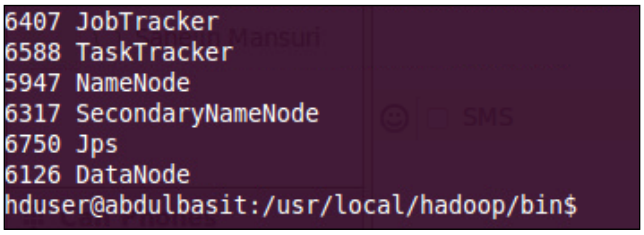


The preceding screenshot shows all the components which have started; they're listed one by one.

Once started, you need to check whether all the components are running perfectly or not. For that, run the following command:

```
hduser@ubuntu:/usr/local/hadoop$ jps
```

If all succeeds, you will get the following output:



The preceding screenshot shows the number of components running in Apache Hadoop. You can always refer to `http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-Win-1.3.0/bk_getting-started-guide/content/ch_hdp1_getting_started_chp2_1.html` for any additional information:

- `JobTracker`: This is a component that will keep track of the number of jobs running in Apache Hadoop and divide each job into the number of tasks that are performed by `TaskTracker`.

- `TaskTracker`: This is used for performing tasks given by `JobTracker`. So, each task tracker has multiple tasks to be performed. And once it is completed with a particular task, it will inform the `JobTracker`. That's how `JobTracker` will get an update that tasks are being performed in the desired manner.
- `Namenode`: This keeps track of the directories created inside HDFS. You can iterate to those directories using `Namenode`. The responsibility of `Namenode` is to transfer data to `Datanodes`. So, `Namenode` itself will not store any data. Rather, it will transfer all the data to the `DataNode`.
- `SecondaryNameNode`: This is a backup of `Namenode`. So, whenever any `Namenode` fails, we can back up our data from `SecondaryNamenode`.
- `DataNode`: This is the component which actually stores the data transferred from `NameNode`. So, the responsibility of `DataNode` is to store all the data of Apache Hadoop.
- `jps`: This is not an Apache Hadoop component. It's a command that is a part of Sun Java since v1.5.0.

Just check on your browser by hitting the following URL:

```
http://localhost:50070
```

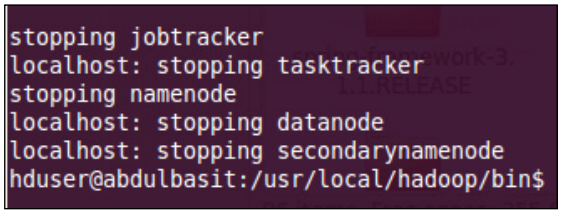
where, `50070` is the port on which `namenode` is running.

Stopping your single-node cluster

If you want to stop your running cluster, hit the following command. Make sure you are logged in as a `hduser` before hitting this command.:

```
hduser@ubuntu:~$ /usr/local/hadoop/bin/stop-all.sh
```

If all succeeds, you will get an output as follows:



The preceding screenshot shows the number of components in Apache Hadoop that are being stopped.

So that's all for installation of Apache Hadoop on a single machine. Now, we will move over to setting up the deployment architecture of Apache Nutch.

Setting up the deployment architecture of Apache Nutch

We have to set up Apache Nutch on each of the machines that we are using. In this case, we are using a six-machine cluster. So, we have to set up Apache Nutch on all of the machines. If there are a small number of machines in our cluster configuration, we can set it up manually on each machine. But when there are more machines, let's say we have 100 machines in our cluster environment, we can't set it up on each machine manually. For that, we require a deployment tool such as **Chef** or at least distributed SSH. You can refer to `http://www.opscode.com/chef/` for getting familiar with Chef. You can refer to `http://www.ibm.com/developerworks/aix/library/au-satdistadmin/` for getting familiar with distributed SSH. I will just demonstrate running Apache Hadoop on Ubuntu for a single-node cluster. If you want to run Apache Hadoop on Ubuntu for a multi-node cluster, I have already provided the reference link. You can follow that and configure it from there. Once we are done with the deployment of Apache Nutch to a single machine, we will run this `start-all.sh` script that will start the services on the master node and data nodes. This means that the script will begin the Hadoop daemons on the master node. So, we are able to login to all the slave nodes using the SSH command as explained, and this will begin daemons on the slave nodes.

The `start-all.sh` script expects that Apache Nutch should be put on the same location on each machine. It is also expecting that Apache Hadoop is storing the data at the same file path on each machine. The `start-all.sh` script that starts the daemons on the master and slave nodes are going to use password-less login by using SSH.

Installing Apache Nutch

Download Apache Nutch from `http://nutch.apache.org/downloads.html` and extract the contents of the Apache Nutch package to your preferred location. I picked the `/usr/local` directory for Apache Nutch. You need to assign permission to Apache Nutch so that only `hduser` can access it. This is for security purposes. The commands that will be used are given as follows:

- The following command will take you to the `local` directory:

  ```
  $ cd /usr/local
  ```

- The following command will be used for extracting the contents of Apache Nutch:

  ```
  $ sudo tar xzf apache-nutch-1.4-bin.tar.gz
  ```

- The following command will rename `apache-nutch-1.4-bin.tar.gz` to `nutch`:

  ```
  $ sudo mv apache-nutch-1.4-bin.tar.gz nutch
  ```

- The following command will assign permission to the `nutch` directory that can be accessed only by `hduser`:

  ```
  $ sudo chown -R hduser:hadoop nutch
  ```

- Now, we need to modify the `bashrc` file. To open this file, go to the `root` directory from your terminal. Then, hit the following command:

  ```
  gedit ~/.bashrc
  ```

  Put the following configuration at the end of the file. It will set your classpath for Apache Nutch.

  ```
  export NUTCH_HOME=/usr/local/nutch
  export PATH=$PATH:$NUTCH_HOME/bin
  ```

- Modify `nutch-site.xml` that you will find in `$NUTCH_HOME/conf` by inserting the following content:

  ```
  <property>
  <name>plugin.folders</name>
  <value>/usr/local/nutch/build/plugins</value>
  </property
  ```

  Search the `http.agent.name` key in `nutch-site.xml`, and set its value to the crawler name.

- Copy `hadoop-env.sh`, `core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`, master, and slaves from `$HADOOP_HOME/conf` to `$NUTCH_HOME/conf` by hitting the following command:

  ```
  $ cd $HADOOP_HOME/conf
  $ cp hadoop-env.sh core-site.xml hdfs-site.xml mapred-site.xml
  master slaves $NUTCH_HOME/nutch/conf/
  ```

- Copy the `conf` directory from `$NUTCH_HOME` to `$NUTCH_HOME/runtime/local/conf` by hitting the following command:

  ```
  $ cd $NUTCH_HOME
  $ cp conf/* runtime/local/conf/
  ```

Key points of the Apache Nutch installation

We need to rebuild Apache Nutch by using the `ant` command. Otherwise it will give a fatal error as `http.agent.name` doesn't work even when we edit the `nutch-site.xml` file. And, we also need to set the `classpath` in `hadoop-env.sh`, which you will find in `$HADOOP_HOME/conf` by putting the following configuration into it:

```
Hadoop_classpath=/usr/local/nutch/runtime/lib/*:/usr/local/nutch/
runtime/deploy/apache-nutch-2.2.1.job
```

After this, go to the `$NUTCH_HOME` directory and type the following command for rebuilding Apache Nutch:

```
/usr/local/nutch/$ ant
```

Once rebuild is finished, copy the `nutch-2.2.1.job` and `nutch-2.2.1.jar` to the `deploy` and `local` directory of Apache Nutch respectively, by typing the following command:

```
$ cp $NUTCH_HOME/build/nutch-2.2.1.job $NUTCH_HOME/runtime/deploy
$ cp $NUTCH_HOMEbuild/nutch-2.2.1.jar
$NUTCH_HOME/runtime/local/lib/
```

We have successfully set up Apache Nutch on the Apache Hadoop cluster. Now, we can start Apache Hadoop and perform the tasks of Apache Nutch on that.

Starting the cluster

We've configured Apache Hadoop. It's time to start up Apache Hadoop on a single node and check whether it's working properly or not. To start up all of the Hadoop components on the local machine (`NameNode`, `DataNode`, `TaskTracker`, `JobTracker`, and `SecondaryNameNode`), the following command will be used:

```
HADOOP_HOME$bin/start-all.sh
```

If you want to stop all components, the following command should be used:

```
HADOOP_HOME$bin/stop-all.sh
```

Performing crawling on the Apache Hadoop cluster

We are going to perform crawling in Apache Nutch on the Apache Hadoop Cluster. It will perform crawling on the Apache Hadoop cluster and give us the result. The steps for performing this job are given as follows:

1. Create the `seed.txt` file by hitting the following commands. This file will contain the list of URLs to be crawled on the Apache Hadoop Cluster.

   ```
   $cd $NUTCH_HOME/runtime
   $mkdir urlsdir
   $ vi urls/seed.txt
   ```

2. The preceding command will create the `seed.txt` file and open it up. Put the following configuration into it:

   ```
   http://nutch.apache.org
   http://apache.org
   ```

   The preceding URLs are used for crawling. You can enter *n* number of URLs, but one URL per line.

3. Now we need to add the `urls` directory to the HDFS directory as Apache Hadoop will use this directory for crawling. The following command will be used for adding this directory. Make sure you are logged in as an `hduser` before hitting the following command:

   ```
   HADDOP_HOME$bin/hadoop dfs -put
   /usr/local/nutch/runtime/local/urls urls
   ```

4. For checking whether it's correctly put or not, type the following command. It will list all the directories that are inside the given directory.

   ```
   $bin/hadoop dfs –ls
   ```

5. Modify `regex-urlfilter.txt`, which sets the filter to crawl only the webpages as `*.apache.org`. So, any web URL that ends with `apache.org` will be crawled. Fire the following commands to perform this:

   ```
   $cd $NUTCH_HOME/runtime
   $ vi conf/regex-urlfilter.txt
   ```

6. The preceding command will open up `regex-urlfilter.txt`. Replace the line `+^http://([a-z0-9]*\.)*MY.DOMAIN.NAME/` with `+^http://([a-z0-9]*\.)*apache.org/`.

We have added our `urls` directory to the Hadoop distributed filesystem and we also have edited our `regex-urlfilter.txt`. Now, it's time to start crawling. To begin with Apache Nutch crawling, first copy your `apache-nutch-2.2.1.job jar` file (you will find in `$NUTCH_HOME/build`) to `$HADOOP_HOME`. Then use the following command to perform crawling. Make sure that you are logged in as an `hduser` before hitting the following commands:

- The following command will take you to the `HADOOP_HOME` directory:

  ```
  $cd $HADOOP_HOME
  ```

- The following commands will actually perform crawling:

  ```
  $hadoop jar apache-nutch-${version}.job
  org.apache.nutch.crawl.Crawl urls -dir crawl -depth 3 -topN5
  ```

  - `hadoop`: This is a command for running Hadoop.
  - `jar`: This is a command for defining the JAR files.
  - `apache-nutch-${version}.job`: This is an Apache Nutch job jar file, which is used for crawling.
  - `apache.nutch.crawl.Crawl`: This is a command for crawling.
  - `urls`: This is a directory that Apache Hadoop will use for fetching the URLs that will be crawled. This is the directory which we put in the HDFS.
  - `dir`: This is a command for defining the directory.
  - `crawl`: This is the actual directory where Apache Hadoop keeps the output of crawling.
  - `depth`: This is a command that is used for setting the number of iterations Apache Nutch will take to crawl. You can give any integer value to it.
  - `topN`: This is a command that is used by Apache Nutch to define the number of top-most URLs that need to be crawled. So, only those URLs will be crawled from the `urls` directory. You can give any integer value to it, but with the condition that `topN` should be greater than or equal to the total number of URLs in the `urls` directory.

So, there are many arguments which you can apply according to your needs.
If all succeeds, you will get the following output:

```
13/10/17 13:27:29 INFO mapred.JobClient:     Map output records=0
13/10/17 13:27:29 INFO crawl.LinkDb: LinkDb: finished at 2013-10-17 13:27:29, elapse
13/10/17 13:27:29 INFO crawl.Crawl: crawl finished: crawl
hduser@reeeshu:/usr/local/hadoop/bin$
```

The preceding screenshot is showing that Apache Hadoop crawled URLs for you. Its showing the last three lines of the output. You can also keep track of your crawling from the browser by opening the `Jobtracker` component of Apache Hadoop. It will show you the statistics of the running jobs, and also the number of tasks per job. Hit the following URL to check this:

```
http://localhost:50030/jobtracker.jsp
```

If all succeeds, you will get an output as follows:

Completed Jobs

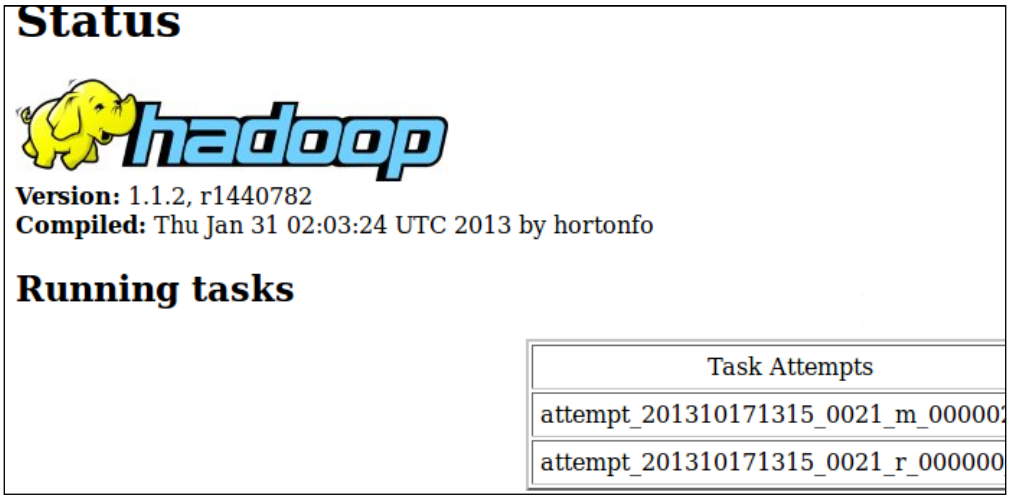
Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total
job_201310171315_0002	Thu Oct 17 13:20:34 IST 2013	NORMAL	hduser	inject urls	100.00%	2	2	100.00%	1
job_201310171315_0003	Thu Oct 17 13:20:57 IST 2013	NORMAL	hduser	crawldb crawl/crawldb	100.00%	1	1	100.00%	1
job_201310171315_0004	Thu Oct 17 13:21:15 IST 2013	NORMAL	hduser	generate: select from crawl/crawldb	100.00%	1	1	100.00%	1

The preceding screenshot will show you the number of jobs and their statuses for the crawling that you have performed.

You can also check the detailed statistics of your tasks per job by opening the `Tasktracker` component of Apache Hadoop. Hit the following URL to check this:

```
http://localhost:50060/tasktracker.jsp
```

If all succeeds, you will get an output as follows:



The preceding screenshot is showing the task tracker that will keep track of the number of tasks for the particular job.

You could take the dump of the crawled URLs by copying the directory from HDFS, which contains data about all the crawled URLs. Copy that directory and paste it to your preferred location for backup. So, now we have completed integration of Apache Nutch with Apache Hadoop. We have successfully crawled URLs in Apache Nutch on Apache Hadoop Cluster. Now we will move over to Apache nutch configuration with eclipse in the next section. So let's see what it is.

Configuring Apache Nutch with Eclipse

In this section, we will see how we can integrate Apache Nutch with Eclipse. So, just like we performed crawling in Apache Nutch using the command line, we can perform crawling using Java API.

Introducing Apache Nutch configuration with Eclipse

Apache Nutch can be easily configured with Eclipse. After that, we can perform crawling easily using Eclipse. So, we need to perform crawling from the command line. We can use eclipse for all the operations of crawling that we are doing from the command line. Instructions are provided for fixing a development environment for Apache Nutch with Eclipse IDE. It's supposed to give a comprehensive starting resource for configuring, building, crawling, and debugging of Apache Nutch.

The prerequisites for Apache Nutch integration with Eclipse are given as follows:

- Get the latest version of Eclipse from `http://www.eclipse.org/downloads/packages/release/juno/r`
- All the required components are available from them Eclipse Marketplace. You can download the from this link: `http://marketplace.eclipse.org/marketplace-client-intro`
- Once you've configured Eclipse, download as per here `http://subclipse.tigris.org/`.

If you have faced a problem with the 1.8.x release, try 1.6.x. This may resolve compatibility issues.

- Download the IvyDE plugin for Eclipse from the following link: `http://ant.apache.org/ivy/ivyde/download.cgi`
- Download the m2e plugin for Eclipse from the following link: `http://marketplace.eclipse.org/content/maven-integration-eclipse`

Installation and building Apache Nutch with Eclipse

Here, we will define the installation steps for configuring Apache Nutch with Eclipse. The steps for configuring Apache Nutch with Eclipse are given as follows:

1. Get the latest source code of Apache Nutch by using SVN, which is a subversion repository. Go to your terminal and fire the following commands:
    - For Apache Nutch 2.2.1, the following command will be used:

      ```
      $svn co https://svn.apache.org/repos/asf/nutch/trunk
      $ cd trunk
      ```

    - For Apache Nutch 2.x, the following command will be used:

      ```
      $svn co https://svn.apache.org/repos/asf/nutch/branches/2.x
      $ cd 2.x
      ```

2. You need to decide which data store you are going to use for this integration. See Apache Gora Documentation from `http://gora.apache.org/` for more information. Some of the choices of storage classes are given as follows:
    - `org.apache.gora.hbase.store.HBaseStore`
    - `org.apache.gora.cassandra.store.CassandraStore`
    - `org.apache.gora.accumulo.store.AccumuloStore`
    - `org.apache.gora.avro.store.AvroStore`
    - `org.apache.gora.avro.store.DataFileAvroStore`

3. Modify `nutch-site.xml`, which you will find in `<Respected directory where your Apache Nutch resides>/conf`, by putting the following configuration into it. I am taking `Hbase` as the `datastore` here.

    ```
    <property>
      <name>storage.data.store.class</name>
      <value>org.apache.gora.hbase.store.HBaseStore</value>
      <description>Default class for storing data</description>
     </property>
    ```

4. Modify `ivy.xml`, which you will find in `<Respected directory where your Apache Nutch resides>/ivy`, by uncommenting the following line if you have taken `hbase` as the data store. Otherwise, you have to search for your datastore and uncomment it if it is commented accordingly.

    ```
    <dependency org=""org.apache.gora"" name=""gora-hbase""
    rev=""0.3"" conf=""*->default"" />
    ```

5. Modify `gora.properties`, which you will find in `<Respected directory where your Apache Nutch resides>/conf`, by in putting the following line if you have taken Hbase as your datastore. Otherwise you have to find out that line for your datastore and in put it accordingly.

   ```
   gora.datastore.default=org.apache.gora.hbase.store.HBaseStore
   ```

6. Modify `nutch-site.xml` by putting `http.agent.name` (as explained before) and `http.robots.agent` (same as `http.agent.name`) into it. Also, set the `plugin.folders` property by putting the following configuration into it:

   ```
   <property>
      <name>plugin.folders</name>
      <value>/home/tejas/Desktop/2.x/build/plugins</value>
    </property>
   ```

   The value of the `plugin.folders` would be `<Respected directory where your Apache Nutch resides>/build/plugins`.

7. Run the following command for building Eclipse:

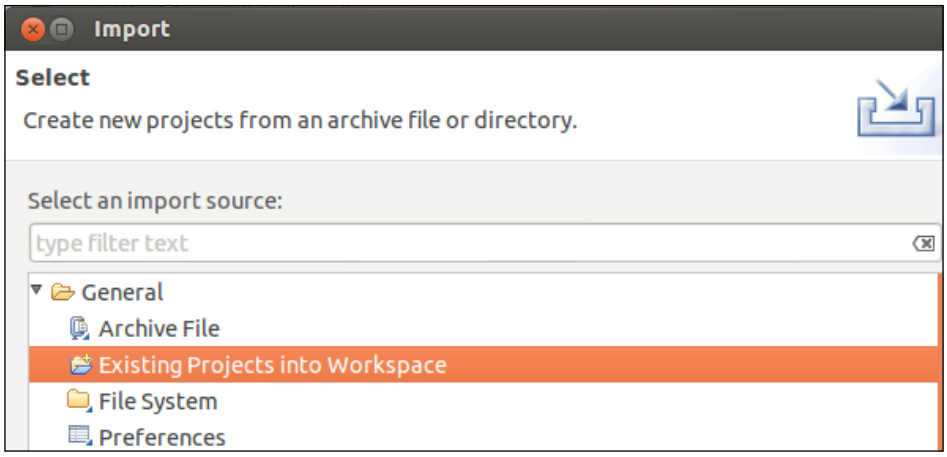
   ```
   $cd <Respected directory where your Apache Nutch resides>
   $ant eclipse
   ```

We have configured this successfully. Now, we shall move to the *Crawling in Eclipse* section.

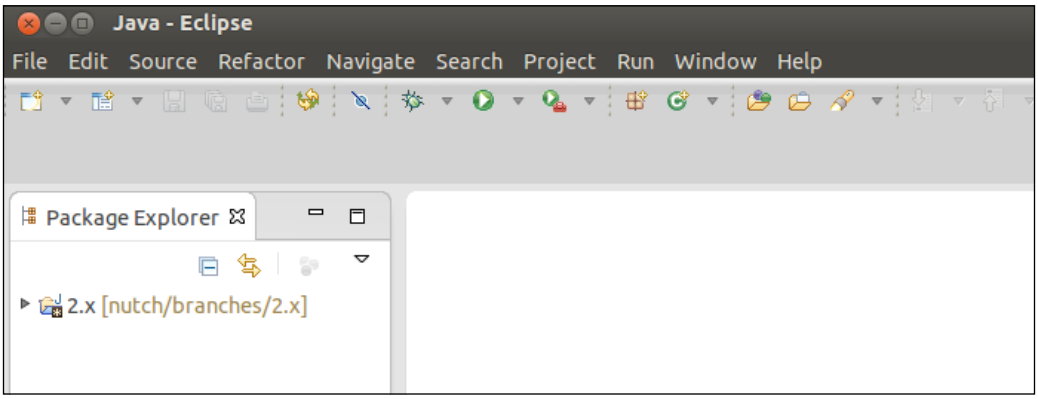
Crawling in Eclipse

In this section, we will see how we can import Apache Nutch into Eclipse and perform crawling. The steps are given as follows:

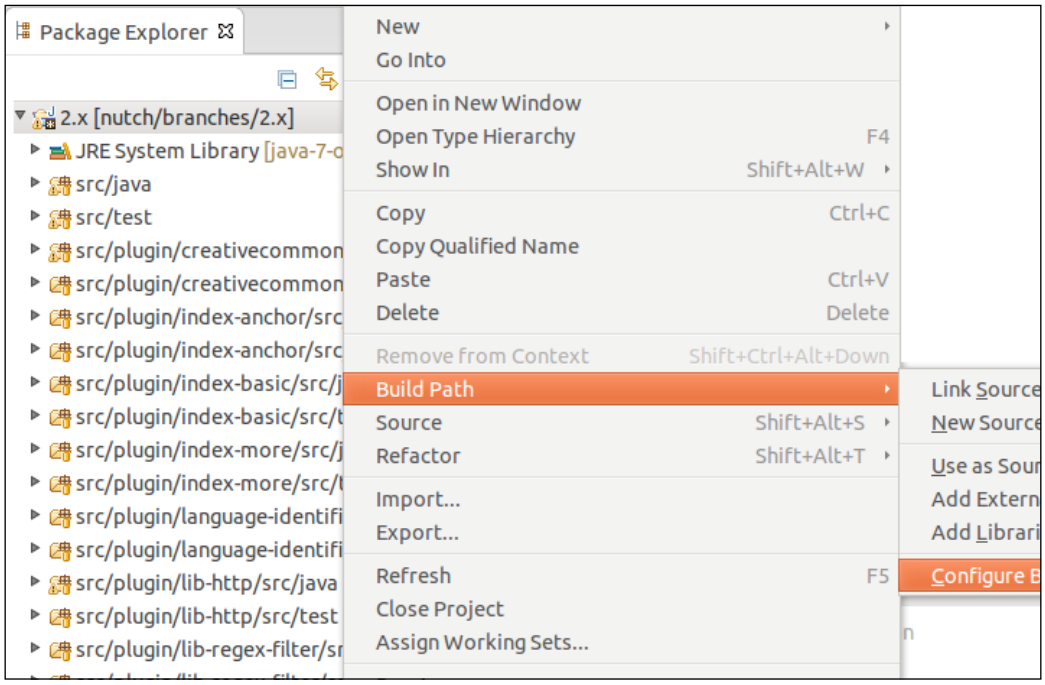
1. Open your Eclipse IDE.
2. In Eclipse, navigate to **File** | **Import**.
3. Select **Existing Projects into Workspace** as shown in the following screenshot:



4. In the next window, set the `root` directory to the place where you have done the checkout of Apache Nutch 2.1.1, and then click on **Finish**.
5. You are currently seeing a new project named **2.1.1**, which is being added within the workspace. Wait for some time until Eclipse refreshes its SVN cache and builds its workspace. You'll get the status at the end point of the corner of the Eclipse window, as shown in the next screenshot:



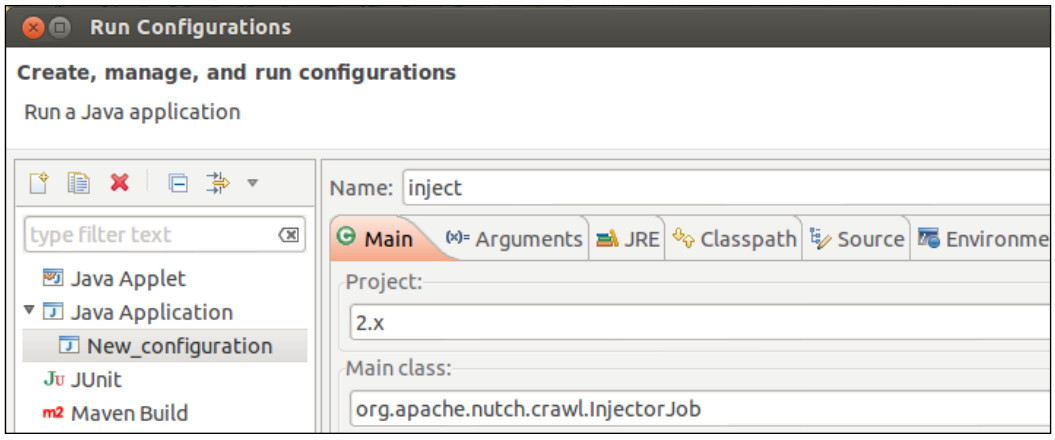
6. In the **Package Explorer**, right-click on the project **2.1.1** and navigate to **Build Path** | **Configure Build Path** as shown below:



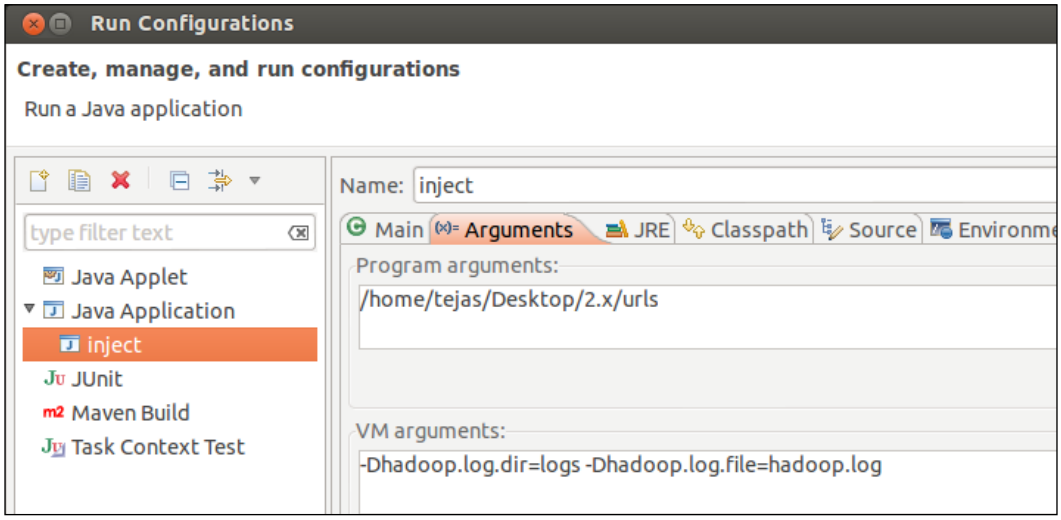
7. In the **Order and Export** tab, scroll down and choose **2.x/conf**. Click on the **Top** button. Sadly, Eclipse will take one more build of the workspace, but this time it won't take much time.

How to create an Eclipse launcher? Let's start with the inject operation. The steps for this are given as follows:

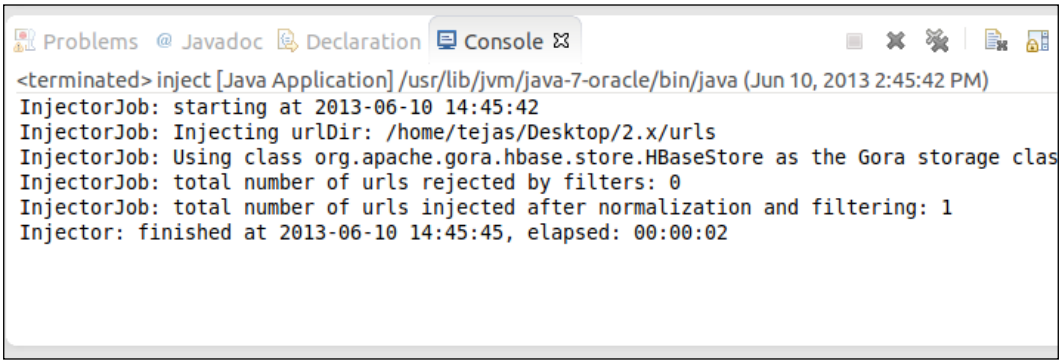
1. Right-click on the project by navigating to **Package Explorer** | **Run As** | **Run Configurations**. Make a new configuration. Name it `inject`, as shown below:
    - **For Apache version 1.x**: Set the main class value as `org.apache.nutch.crawl.Injector`
    - **For Apache version 2.x**: Set the main class as `org.apache.nutch.crawl.InjectorJob`



2. In the **Arguments** tab for program arguments, give the path of the input `dir`, which has the seed URLs. Set VM Arguments to `-Dhadoop.log.dir=logs -Dhadoop.log.file=hadoop.log`, as shown in the following screenshot:



3. Click on **Apply** and then click on **Run**. If everything was done perfectly, then you will see the inject operation progressing on the console as shown in the following screenshot:



If you want to find out the Java class related to any command, just go inside the `src/bin/nutch` script; at the bottom, you will find a `switch-case` code with a `case` corresponding to each command. The important classes corresponding to the `crawl` cycle are given as follows:

Operation	Class in Nutch 1.x (that is, trunk)	Class in Nutch 2.x
inject	`org.apache.nutch.crawl.Injector`	`org.apache.nutch.crawl.InjectorJob`
generate	`org.apache.nutch.crawl.Generator`	`org.apache.nutch.crawl.GeneratorJob`
fetch	`org.apache.nutch.fetcher.Fetcher`	`org.apache.nutch.fetcher.FetcherJob`
parse	`org.apache.nutch.parse.ParseSegment`	`org.apache.nutch.parse.ParserJob`
updated	`org.apache.nutch.crawl.CrawlDb`	`org.apache.nutch.crawl.DbUpdaterJob`

In the same way, you can perform all the jobs that are listed in the preceding table. You can take the respective Java class of a particular job and run that within Eclipse. So that's how crawling occurs in Apache Nutch using Eclipse.

So, now we have successfully integrated Apache Nutch with Eclipse. So that's the end of this chapter. Let's go to the *Summary* section now and revise what you have learned from this chapter.

Summary

We started with the integration of Apache Nutch with Apache Hadoop. In that, we have covered an introduction to Apache Hadoop, what do we mean by integrating Apache Nutch with Apache Hadoop, and what are the benefits of that. Then, we moved on to the configuration steps, and we configured Apache Nutch with Apache Hadoop successfully. We also performed a crawling job by installing Apache Nutch on a machine, and confirmed the output—Apache Hadoop cluster is running properly and is performing the crawling job correctly. Then, we started with integration of Apache Nutch with Eclipse. We also had a little introduction to what is integration of Apache Nutch with Eclipse. Then, we looked at the configuration of Apache Nutch with Eclipse. We have successfully integrated Apache Nutch with Eclipse and performed one InjectorJob example.

I hope you have enjoyed reading this chapter. Now, let's see how we can integrate Apache Nutch with Gora, Accumulo, and MySQL in the next chapter.

4
Apache Nutch with Gora, Accumulo, and MySQL

This chapter covers the integration of Apache Nutch with Gora, Accumulo, and MySQL. Gora, Accumulo, and MySQL are mainly used as databases. This chapter covers each of these in detail. First, we will start with the integration of Apache Nutch with Apache Accumulo using Gora and then we will move on to the integration of Nutch with MySQL using Gora.

In this chapter we will cover the following topics:

- Introduction to Apache Accumulo
- Introduction to Apache Gora
- Integration of Nutch with Apache Accumulo
    - Configuring Gora with Apache Nutch
    - Setting up Hadoop and Apache ZooKeeper
    - Installing and configuring Accumulo
    - Testing Apache Accumulo
    - Crawling with Apache Nutch on Apache Accumulo
- Integration of Nutch with MySQL
    - Introduction to MySQL
    - Need for integrating MySQL with Nutch
    - Configuring MySQL with Nutch
    - Crawling with Nutch on MySQL

By the end of this chapter, you will be able to integrate Nutch with Apache Accumulo as well as with MySQL. After that, you can perform crawling using Apache Nutch on Apache Accumulo and also on MySQL, and you can get the results of your crawled pages on Accumulo as well as on MYSQL. You can also perform the integration of Apache Solr as discussed earlier and get your crawled pages indexed onto the Apache Solr.

Introduction to Apache Accumulo

Accumulo is basically used as the data store for storing data the same way we use different databases such as MySQL and Oracle. The key point of Apache Accumulo is that it runs on the Apache Hadoop Cluster environment, which is a very good feature of Accumulo. With Accumulo sorted, the distributed key/value store could be a strong, scalable, high-performance information storage and retrieval system. Apache Accumulo depends on Google's BigTable design and is built on top of Apache Hadoop, Thrift, and ZooKeeper. Apache Accumulo features some novel improvement on the BigTable design in the form of cell-based access management and the server-side programming mechanism, which will perform modification in the key/value pairs at varied points within the data management process.

Main features of Apache Accumulo

The following are the two main features of Accumulo:

- **Security at cell level**: Apache Accumulo extends a BigTable data model, adding a new component called **Column Visibility** to the key. This component stores a logical combination of security labels that need to be satisfied at query time in order for the key and value to be returned as a part of the user request. This enables data of varying security levels to be stored within the same table and allows users to see only those keys and values for which they are authorized.
- **Programming at the server side**: In addition to the security at the cell level, Apache Accumulo provides a programming mechanism at the server side called **Iterators**, which enables users to perform extra process at the table server. The range of operations that may be applied is equivalent to those that can be implemented within a MapReduce Combiner function, which produces an aggregate value for several key/value pairs.

Introduction to Apache Gora

Apache Gora's open source framework provides an in-memory data model and persistence for large data. Apache Gora supports persisting to column stores, key-value stores, document stores, and RDBMS, and analyzing the data with extensive Apache Hadoop MapReduce support.

Supported data stores

Apache Gora presently supports the following data stores:

- Accumulo Prophets
- Apache Hbase
- Amazon DynamoDB

Use of Apache Gora

Although there are many excellent ORM frameworks for relational databases and data modeling, data stores in NoSQL are profoundly different from their relative cousins. Data model agnostic frameworks such as JDO aren't easy for use cases, where one has to use the complete power of data models in column stores. Gora fills the gap giving users an easy-to-use in-memory data model plus persistence for large data framework, providing data store-specific mappings and also in-built Apache Hadoop support.

Integration of Apache Nutch with Apache Accumulo

In this section, we are going to cover the integration process for integrating Apache Nutch with Apache Accumulo. Apache Accumulo is basically used for a huge data storage. It is built on the top of Apache Hadoop, ZooKeeper, and Thrift. So, a potential use of integrating Apache Nutch with Apache Accumulo is when our application has huge data to process and we want to run our application in a cluster environment. At such times, we can use Apache Accumulo for data storage purposes. As Apache Accumulo only runs with Apache Hadoop, the maximum use of Apache Accumulo would be in a cluster-based environment. We will first start with the configuration of Apache Gora with Apache Nutch. Then, we will set up Apache Hadoop and ZooKeeper. After this, we will do the installation and configuration of Apache Accumulo. Following this, we will test Apache Accumulo, and at the end we will see the process of crawling with Apache Nutch on Apache Accumulo.

Configuring Apache Gora with Apache Nutch

In this section, we are going to configure Apache Gora with Apache Nutch.

I have used Nutch 2.2.1.

So, you have to get Apache Nutch as explained before. Here, you have to configure Apache Gora for integration of Apache Nutch with Apache Accumulo. The following are the steps for doing this:

1. Modify the `ivy.xml` file that you will find in `$NUTCH_HOME`, which will be the path where your Apache Nutch resides. Change revised versions of `gora-core` and `gora-sql` dependencies from 0.1.1, incubating to `"0.2-SNAPSHOT"`. Also, add the subsequent lines:

```
<dependency org="org.apache.accumulo" name="accumulo-core"
rev="1.5.0" />
<dependency org="org.apache.accumulo" name="cloudtrace"
rev="1.4.0" />
<dependency org="org.apache.thrift" name="libthrift" rev="0.6.1"
/>
<dependency org="org.apache.gora" name="gora-accumulo" rev="0.2"
/>
<dependency org="org.apache.zookeeper" name="zookeeper"
rev="3.4.3" />
```

2. The preceding configuration defines all the necessary components that are used for this integration. Modify the `ivysettings.xml` file that you will find in `$$NUTCH_HOME/ivy`. It will configure `ant/ivy` to use your local Maven repository and resolving dependencies. This is required because the patched version of Gora and the latest Accumulo version are not present in any public Maven repository. Add the following lines under the `<resolvers>` section at the top:

```
<ibiblio name="local"root="file://${user.home}/.m2/repository/"
  pattern="${maven2.pattern.ext}"m2compatible="true"/>
```

   This configuration will set your Maven repository. By default, your Maven repository resides inside your home directory. That's why I have given the path in the `root` property. If you have changed the path, you will have to make changes in the directory accordingly.

3. Patch the following code into `StorageUtils.java`, which you will find in `<Respected directory where your Apache Nutch resides>/src/java/org/apache/nutch/storage/`:

```
diff --git a/src/java/org/apache/nutch/storage/StorageUtils.java
b/src/java/org/apache/nutch/storage/StorageUtils.java
index de740b5..19b37ad 100644
--- a/src/java/org/apache/nutch/storage/StorageUtils.java
+++ b/src/java/org/apache/nutch/storage/StorageUtils.java
@@ -40,8 +40,9 @@ public class StorageUtils {
       Class<K> keyClass, Class<V> persistentClass) throws
ClassNotFoundException, GoraException {
     Class<? extends DataStore<K, V>> dataStoreClass =
       (Class<? extends DataStore<K, V>>) getDataStoreClass(conf);
+
     return DataStoreFactory.createDataStore(dataStoreClass,
-            keyClass, persistentClass);
+            keyClass, persistentClass, conf);
   }

   @SuppressWarnings("unchecked")
@@ -56,8 +57,9 @@ public class StorageUtils {

     Class<? extends DataStore<K, V>> dataStoreClass =
       (Class<? extends DataStore<K, V>>) getDataStoreClass(conf);
+
     return DataStoreFactory.createDataStore(dataStoreClass,
-            keyClass, persistentClass, schema);
+            keyClass, persistentClass, conf, schema);
   }

   @SuppressWarnings("unchecked")
```

The preceding configuration will configure `StorageUtils.java`. This is the class that will process the data crawled by Apache Nutch and store it inside the table.

4. Create the `gora-accumulo-mapping.xml` file in `<Respected directory where your Apache Nutch resides>/conf` and include the following content:

```
<gora-orm>
<table name="webpage">
<config key="table.file.compress.blocksize" value="32K"/>
</table>
```

```
<class table="webpage" keyClass="java.lang.String"
name="org.apache.nutch.storage.WebPage">
<!-- fetch fields -->
<field name="baseUrl" family="f" qualifier="bas"/>
<field name="status" family="f" qualifier="st"/>
<field name="prevFetchTime" family="f" qualifier="pts"/>
<field name="fetchTime" family="f" qualifier="ts"/>
<field name="fetchInterval" family="f" qualifier="fi"/>
<field name="retriesSinceFetch" family="f" qualifier="rsf"/>
<field name="reprUrl" family="f" qualifier="rpr"/>
<field name="content" family="f" qualifier="cnt"/>
<field name="contentType" family="f" qualifier="typ"/>
<field name="protocolStatus" family="f" qualifier="prot"/>
<field name="modifiedTime" family="f" qualifier="mod"/>
<!-- parse fields -->
<field name="title" family="p" qualifier="t"/>
<field name="text" family="p" qualifier="c"/>
<field name="parseStatus" family="p" qualifier="st"/>
<field name="signature" family="p" qualifier="sig"/>
<field name="prevSignature" family="p" qualifier="psig"/>
<!-- score fields -->
<field name="score" family="s" qualifier="s"/>
<field name="headers" family="h"/>
<field name="inlinks" family="il"/>
<field name="outlinks" family="ol"/>
<field name="metadata" family="mtdt"/>
<field name="markers" family="mk"/>
</class>
</gora-orm>
```

The preceding configuration will create a table called webpage, which is used for storing your data crawled by Apache Nutch. This table will be automatically created at the time of crawling. Whatever fields are defined by the `<field>` tag, those many number of fields will be created with the same name provided in the `name` attribute.

5. Now, we need to edit `gora.properties` by adding the following lines at the end of the file:

```
gora.datastore.default=org.apache.gora.accumulo.store.
AccumuloStore
gora.datastore.accumulo.mock=false
gora.datastore.accumulo.instance=accumulo
```

```
gora.datastore.accumulo.zookeepers=localhost
gora.datastore.accumulo.user=root
gora.datastore.accumulo.password=root
gora.datastore.accumulo.zookeepers=127.0.0.1:2181
```

An explanation of the preceding configuration is as follows:

- `gora.datastore.default` will set up `AccumuloStore` as the default datastore
- `gora.datastore.accumulo.instance` will create an instance of Apache Accumulo
- `gora.datastore.accumulo.zookeepers` will set up the host for ZooKeeper
- `gora.datastore.accumulo.user` will set up the username of Apache Accumulo for accessing databases
- `gora.datastore.accumulo.password` will set up the password of Apache Accumulo for accessing databases
- `gora.datastore.accumulo.zookeepers` will set a port for hosts and a port for ZooKeepers

6. Edit `nutch-site.xml`, which you will find under `<Respected directory where your Apache Nutch resides>/conf` by adding the following properties:

```
<configuration>
<property>
<name>storage.data.store.class</name>
<value>org.apache.gora.accumulo.store.AccumuloStore</value>
</property>

<property>
<name>http.agent.name</name>
<value>Nutch</value>
</property>

</configuration>
```

The preceding configuration will set up `AccumuloStore` by defining the `<name>storage.data.store.class</name>` property. It will also set up `http.agent.name` by defining the `<name>http.agent.name</name>` property. This is required by Apache Nutch for crawling.

7. Edit `ivy-1.1.3.xml`, which you will find inside `$HOME/.ivy2/cache/jaxen/jaxen/`, by commenting out the following lines. Here, `$HOME` would be your home directory:

```
<!--
<dependency org="maven-plugins" name="maven-cobertura-
plugin" rev="1.3" force="true" conf="compile-
>compile(*),master(*);runtime->runtime(*)">
      <artifact name="maven-cobertura-plugin" type="plugin"
ext="plugin" conf=""/>
    </dependency>
    <dependency org="maven-plugins" name="maven-
findbugs-plugin" rev="1.3.1" force="true" conf="compile-
>compile(*),master(*);runtime->runtime(*)">
      <artifact name="maven-findbugs-plugin" type="plugin"
ext="plugin" conf=""/>
    </dependency>
-->
```

   The preceding configuration will be commented by default. So just uncomment it. It is a Maven dependency. Hence, Apache Nutch is using this dependency while building the code. If you have not configured it yet, it will give an error at the time of building Apache Nutch.

8. Run the following commands for building Apache Nutch:

```
# cd $NUTCH_HOME
# ant
```

   The preceding command will build Apache Nutch and set up the necessary configuration for this integration. This is compulsory whenever you do any changes in the Apache Nutch configuration.

We have configured Apache Gora successfully. Now, let's move to our next section, which shows how to set up Apache Hadoop and ZooKeeper.

Setting up Apache Hadoop and Apache ZooKeeper

As Apache Hadoop has already been covered earlier, I am not going to cover that again. Apache ZooKeeper is a centralized service, which is used for maintaining configuration information and provides distributed synchronization, naming, and group services. All of these services are used by distributed applications in one manner or another. All these services are provided by the ZooKeeper, so you don't have to write these services from scratch. You can use these services for implementing consensus, management, group, leader election, and presence protocols, and you can also build it for your own requirements.

Apache Accumulo is built on the top of Apache Hadoop and ZooKeeper. So, we must configure Apache Accumulo within Apache Hadoop and Apache ZooKeeper. You can refer to `http://www.covert.io/post/18414889381/accumulo-nutch-and-gora` for any queries related to the setup. The following are the steps for the configuration:

1. Configure and Start Hadoop. We have discussed this already in *Chapter 3*, *Integration of Apache Nutch with Apache Hadoop and Eclipse*. So, I will not cover this part here; instead, I will directly start Apache Hadoop. The command for starting Apache Hadoop is as follows:

   ```
   #usr/local/hadoop/bin$./start-all.sh
   ```

2. Configure and Start ZooKeeper. While you are making an attempt to start Apache ZooKeeper for the first time, the simple thing to do is run it in the standalone mode with the single ZooKeeper server. You can try this on a development machine for example. Apache ZooKeeper requires Java 6 to run, so you have to make sure that you have it installed. There is no need for Cygwin to run Apache ZooKeeper on Windows since Windows versions of Apache ZooKeeper scripts are available. (Windows supports only as a development platform and not as a production platform.)
3. Download ZooKeeper. We can download a stable release of Apache ZooKeeper from the Apache ZooKeeper releases page at `http://zookeeper.apache.org/releases.html`, and unpack it in a suitable location. I have kept it in `/usr/local`. The following command will be used for unpacking and giving permission to ZooKeeper:

   ```
   # cd /usr/local
   # tar zxf zookeeper-x.x.x.tar.gz
   # mv zookeeper-x.x.x zookeeper
   # chown -R hduser:hadoop zookeeper
   ```

The commands are explained as follows:

- `cd /usr/local` will take you inside the `local` directory
- `tar zxf zookeeper-x.x.x.tar.gz` will extract ZooKeeper
- `mv zookeeper-x.x.x zookeeper` will rename `zookeeper-x.x.x` to `zookeeper`
- `chown -R hduser:hadoop zookeeper` will assign the permission to the `zookeeper` directory that only `hduser` can access this directory

4. Setting the `.bashrc` file. To open this file, go to your root directory and type the following command:

```
#gedit ~/.bashrc
```

It will open up your `.bashrc` file. Now, set up a classpath entry inside your `.bashrc` file for Apache ZooKeeper by adding the following configuration. This is required for running Apache ZooKeeper.

```
export ZK_HOME=/usr/local/zookeeper
export PATH=$PATH:$ZK_HOME/bin
```

5. For checking all the files of ZooKeeper, the following commands need to be used:

```
#cd $ZK_HOME/conf
#ZK_HOME/conf#ls
```

6. Create a directory called `zookeeper`. This directory would be used by Apache Hadoop while Apache Nutch performs crawling. The following commands will be used for creating this directory:

```
#mkdir  -p /app/zookeeper
#chown hduser:hadoop /app/zookeeper
```

7. If you want more tightened security, the following command will be used:

```
#chmod 750 /app/zookeeper
```

8. Modify `zoo.cfg`, which you will find in <`Respected directory where your zookeeper resides>/conf/` by adding the following configuration to the file:

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
```

```
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
dataDir=/app/zookeeper

# the port at which the clients will connect
clientPort=2181
maxClientCnxns=100
```

9. Create the `zookeeper` logs directory by entering the following commands:

   ```
   cd zookeeper-3.4.3
   export ZOO_LOG_DIR=$HOME/blogpost/zookeeper-3.4.3/logs
   mkdir $ZOO_LOG_DIR
   ```

10. Edit `zkEnv.sh`, which you will find in `$ZK_HOME/bin/zkEnv.sh`, right after the following lines:

    ```
    ZOOBINDIR=${ZOOBINDIR:-/usr/bin}
    ZOOKEEPER_PREFIX=${ZOOBINDIR}/..
    ```

    And add the following line:

    ```
    export ZOO_LOG_DIR=${ZK_HOME}/logs
    ```

    We have configured Apache ZooKeeper successfully. Now, it's time to start Apache ZooKeeper, which can be done using the following commands:

    ```
    # cd $ZK_HOME
    #./bin/zkServer.sh start
    ```

    If you have configured Apache ZooKeeper and if it has started properly, you will be able to check it using the following command:

    ```
    ./bin/zkCli.sh -server 127.0.0.1:2181
    ```

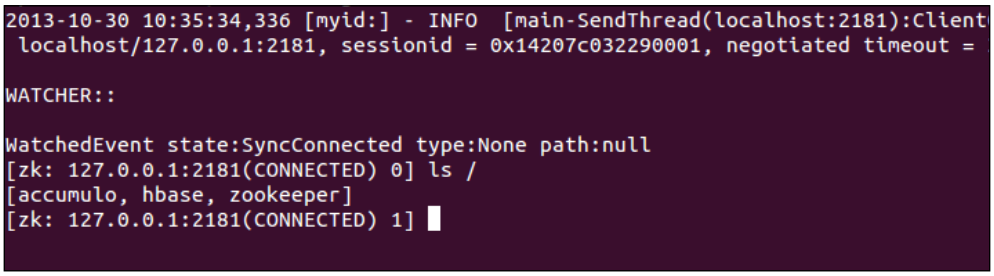
    It will give the output as a bunch of logging messages, which is fine. Press *Enter* and then you must be inside a shell. This will look as follows:

    ```
    [zk: 127.0.0.1:2181(CONNECTED) 0]
    ```

    Then, type `ls /` and press *Enter*. You must see a single line of output (followed again by a prompt) that looks as follows:

    ```
    [zookeeper]
    ```

The following screenshot shows that your ZooKeeper is running successfully:



The preceding screenshot shows only the last lines as the output. If you face any error, you need to check the whole output.

If no errors occur, you have configured Apache ZooKeeper properly. When you run the `zkCli.sh` command for the first time and if you see stack traces as follows:

```
java.net.ConnectException: Connection refused
    at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
    at sun.nio.ch.SocketChannelImpl.
finishConnect(SocketChannelImpl.java:701)
    at org.apache.zookeeper.ClientCnxnSocketNIO.
doTransport(ClientCnxnSocketNIO.java:286)
    at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.
java:1035)
```

This means that Apache ZooKeeper failed to start due to some reason and isn't listening on `127.0.0.1:2181`, or you may have a local firewall blocking access to that port.

Installing and configuring Apache Accumulo

The numbers of tools required for Apache Accumulo are as follows:

- Java (1.6 and higher)
- Apache Hadoop
- Apache ZooKeeper (3.3.3 and higher)

I have used Java Version 1.7.0, Apache Hadoop Version 1.1.2, and ZooKeeper Version 3.3.

The following are the steps for the installation and configuration of Apache Accumulo:

1. Download Apache Accumulo. You can download Apache Accumulo from `http://accumulo.apache.org/downloads/` and unpack it in a suitable location. I have kept Apache Accumulo in the `/usr/local` directory. The following commands will be used for unpacking it and giving access permission to it:

   ```
   $ cd /usr/local
   $ tar zxf accumulo-x.x.x.tar.gz
   $ mv accumulo-x.x.x accumulo
   $ chown -R hduser:hadoop  accumulo
   ```

2. Set the `.bashrc` file. To open this file, go to your `root` directory from the terminal and type the following command:

   ```
   gedit ~/.bashrc
   ```

   It will open up the `.bashrc` file. Now set up a classpath entry in your `.bashrc` file for Apache Accumulo by including the following configuration into it:

   ```
     export ACCUMULO_HOME=/usr/local/accumulo
   export PATH=$PATH:$ACCUMULO_HOME/bin
   ```

3. Create directories for Apache Accumulo with the following commands:

   ```
   mkdir -p /tmp/walogs
   mkdir -p /app/accumulo
   chown hduser:hadoop /app/accumulo
   chmod 750 /app/accumulo
   chown hduser:hadoop /tmp/walogs
   chmod 750 /tmp/walogs
   ```

4. Now we need to copy the `$Accumulo_Home/conf` folder in the `Accumulo` directory, and copy one of the configurations available in the `conf/examples` folder to the `conf` folder with the following commands:

   ```
   #user/local/accumulo/conf$ cp conf/examples/512MB/native-
   standalone/* conf
   ```

5. If you are configuring a larger cluster, you need to create the configuration files yourself and include the changes to the `$ACCUMULO_HOME/conf` directory as follows:
    - Make a `slaves.txt` file in `$ACCUMULO_HOME/conf/` and write `localhost` inside it. This is the configuration on which tablet servers and loggers will run.
    - Create a `masters.txt` file in `$ACCUMULO_HOME/conf/` and write `localhost` in it. This is the configuration of machines where the master server will run.

6. Now create the `accumulo-env.sh` file in `$Accumulo_Home/conf/` by following the template of `$Accumulo_home/conf/example/3GB/native standalone/accumulo-env.sh`, and edit `JAVA_HOME`, `HADOOP_HOME` and `ZOOKEEPER_HOME` inside the `conf/accumulo-env.sh` as follows:

    The following command will increase the memory of Apache Accumulo:

    ```
    test -z "$ACCUMULO_HOME" && export ACCUMULO_HOME=/usr/local/
    accumulo/src/assemble/accumulo-1.5.0/


    test -z "$ACCUMULO_TSERVER_OPTS" && export ACCUMULO_TSERVER_
    OPTS="${POLICY} -Xmx1g -Xms1g -Xss512k"


    test -z "$ACCUMULO_TSERVER_OPTS" && export ACCUMULO_TSERVER_
    OPTS="${POLICY} -Xmx2g -Xms2g -Xss512k"
    ```

    - The following will be the path where Java resides

      ```
      export JAVA_HOME=/usr/lib/jvm/java-7-oracle
      ```

    - The following will be the path where ZooKeeper resides

      ```
      export ZOOKEEPER_HOME=/usr/local/zookeeper
      ```

    - The following will be the path of the Accumulo log directory

      ```
      export ACCUMULO_LOG_DIR=/tmp/walogs/logs
      ```

    The preceding paths would be different for different users based on their system settings.

7. Edit `accumulo-site.xml` in `$Accumulo_home/conf/` by setting up the ZooKeeper servers in the `instance.zookeeper.host` property, and set the `logger.dir.walog property` as follows:

```
<property>
    <name>instance.zookeeper.host</name>
    <value>localhost:2181</value>
    <description>comma separated list of zookeeper servers</
description>
 </property>
<property>
  <name>logger.dir.walog</name>
  <value>$HOME/tmp/walogs</value>
</property>
Set the instance.secret property.
<property>
  <name>instance.secret</name>
  <value>SOME-PASSWORD-OF-YOUR-CHOOSING</value>
</property>
<property>
     <name>instance.dfs.dir</name>
     <value>$HOME/app/accumulo/</value>
</property>
```

At this point we have configured Accumulo successfully. It's time to initialize it and start it. Start Accumulo using the following command:

```
#ACCUMULO_HOME$/bin$./accumulo init
```

It will show the output as follows:

```
23 08:15:26,635 [util.Initialize] INFO : Hadoop Filesystem is
hdfs://127.0.0.1/
23 08:15:26,637 [util.Initialize] INFO : Accumulo data dir is /
accumulo
23 08:15:26,637 [util.Initialize] INFO : Zookeeper server is
localhost:2181


Warning!!! Your instance secret is still set to the default, this
is not secure. We highly recommend you change it.


You can change the instance secret in accumulo by using: bin/
accumulo org.apache.accumulo.server.util.ChangeSecret oldPassword
newPassword.
You will also need to edit your secret in your configuration file
by adding the property
instance.secret to your conf/accumulo-site.xml. Without this
accumulo will not operate correctly
```

```
Instance name : inst
Enter initial password for root: ******
Confirm initial password for root: *****
23 08:15:34,100 [util.NativeCodeLoader] INFO : Loaded the native-
hadoop library
23 08:15:34,337 [security.ZKAuthenticator] INFO : Initialized root
user with
username: root at the request of user !SYSTEM
```

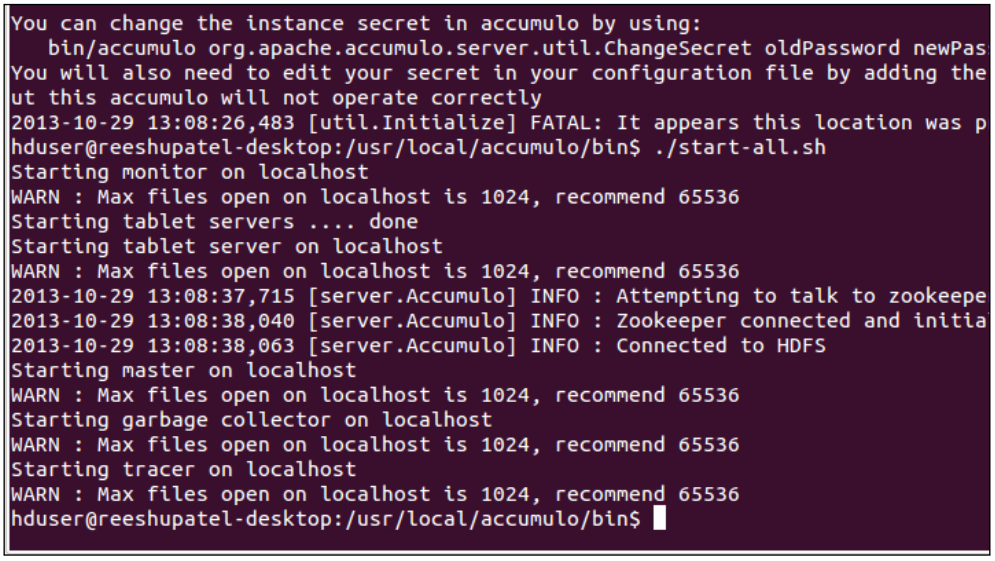
Here, I set my instance name to `inst` and my password to `root`. You can do this in the same way, or make sure that you set the right configuration parameters later.

Testing Apache Accumulo

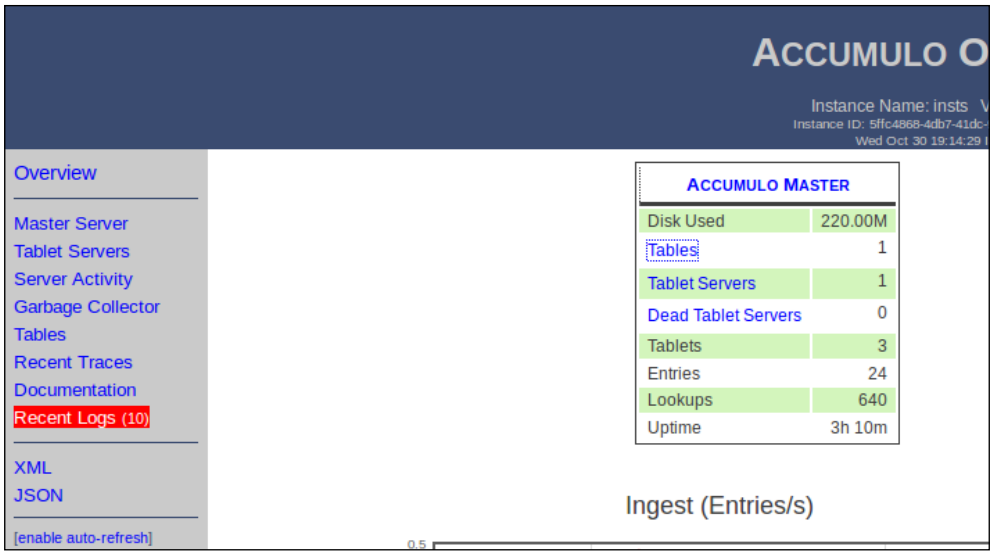
You can start Accumulo using the following command:

```
#ACCUMULO_HOME$./bin/start-all.sh
```

If all succeeds, it will give the output as shown in the following screenshot:



After completing this, you will be able to open `http://127.0.0.1:50095/` in your web browser, and you will see a page as shown in the following screenshot:



The preceding screenshot shows that your Apache Accumulo is running on the browser successfully.

Now, before starting crawling, you have to start the Apache Accumulo server. You can do this with the following command:

```
$ACCUMULO_HOME/bin$./accumulo tserver
```

If all succeeds, you will get the output as shown in the following screenshot:

```
2013-11-01 18:33:07,062 [tabletserver.TabletServer] INFO : port = 9997
2013-11-01 18:33:07,529 [server.Accumulo] WARN : System swappiness setting is
to be delayed.  Accumulo is time sensitive because it needs to maintain distr
2013-11-01 18:33:07,624 [tabletserver.TabletServer] INFO : Loading tablet !0;
2013-11-01 18:33:07,626 [tabletserver.TabletServer] INFO : reeshupatel-deskto
2013-11-01 18:33:07,657 [util.MetadataTable] INFO : Scanning logging entries
2013-11-01 18:33:07,657 [util.MetadataTable] INFO : Getting logs for root tab
2013-11-01 18:33:07,658 [util.MetadataTable] INFO : Returning logs [] for ext
2013-11-01 18:33:07,764 [util.NativeCodeLoader] INFO : Loaded the native-hado
2013-11-01 18:33:07,970 [tabletserver.TabletServer] INFO : Root tablet loaded
2013-11-01 18:36:45,249 [tabletserver.TabletServer] INFO : Adding 1 logs for
2013-11-01 18:41:46,826 [tabletserver.TabletServer] INFO : Adding 1 logs for
2013-11-01 18:43:07,529 [server.Accumulo] WARN : System swappiness setting is
to be delayed.  Accumulo is time sensitive because it needs to maintain distr
2013-11-01 18:46:48,432 [tabletserver.TabletServer] INFO : Adding 1 logs for
2013-11-01 18:51:49,907 [tabletserver.TabletServer] INFO : Adding 1 logs for
2013-11-01 18:53:07,529 [server.Accumulo] WARN : System swappiness setting is
to be delayed.  Accumulo is time sensitive because it needs to maintain distr
```

The preceding screenshot shows that the server in Apache Accumulo is running successfully.

Crawling with Apache Nutch on Apache Accumulo

By this time, you must have a fully functional version of Apache Hadoop, Zookeeper, and Apache Accumulo installed. So, we are ready to run Apache Nutch web crawler. The following are the steps to do this:

1. Make a file called `seed.txt` in `$NUTCH_HOME/local/runtime/urls`. If the `urls` directory isn't created already, you have to create that directory. Put one URL per line as follows:

   ```
   http://projects.apache.org/indexes/alpha.html
   http://www.dmoz.org/Arts/People/
   ```

2. Now, we need to go to the `$NUTCH_HOME` directory and run the following command:

   ```
   #NUTCH_HOME$./runtime/local/bin/nutch crawl file://$NUTCH_HOME/
   seeds.txt -depth 1
   ```

   You may see some log messages print to the console, but hopefully no stack traces. If you are seeing stack traces, you must go back and check your configuration to make sure they match with the ones we made earlier.

3. After the crawler has completed its run, you are able to explore it using the Apache Accumulo shell. To start Apache Accumulo, type the following command:

   ```
   #ACCUMULO_HOME/bin$./accumulo shell -u root -p secret
   ```

   If all succeeds, you will get an output as follows:

```
reeshupatel@reeshupatel-desktop:~$ sudo -s
[sudo] password for reeshupatel:
root@reeshupatel-desktop:~# cd /usr/local/accumulo/src/assemble/accumulo-1
root@reeshupatel-desktop:/usr/local/accumulo/src/assemble/accumulo-1.5.0/b
Password: ****

Shell - Apache Accumulo Interactive Shell
-
- version: 1.5.0
- instance name: reeshu1
- instance id: 38c49e84-0597-481a-be97-8e583958a0a6
-
- type 'help' for a list of available commands
-
root@reeshu1>
```

The preceding screenshot shows the Apache Accumulo shell.

To check the table, you need to look inside the table with the following commands:

```
root@reeshu1> table webpage
root@reeshu1 webpage> scan
```

It will give the output as follows:

```
org.apache.projects:http/categories.html f:fi [] \x00'\x8D\x00
org.apache.projects:http/categories.html f:st [] \x00\x00\x00\x01
org.apache.projects:http/categories.html f:ts []
...
...
...
```

We have successfully integrated Apache Nutch with Apache Accumulo. Now, let's move to our next section that shows how to integrate Apache Nutch with MySQL.

Integration of Apache Nutch with MySQL

In this section, we are going to integrate Apache Nutch with MySQL. After the integration, you can crawl web pages in Apache Nutch that will be stored in MySQL. So, you can go to MySQL and check your crawled web pages and also perform the necessary operations. We will start with the introduction of MySQL and then we will cover the benefits of integrating MySQL with Apache Nutch. After that, we will see the configuration of MySQL with Apache Nutch, and finally, we will perform crawling with Apache Nutch on MySQL. So, let's start with an introduction to MySQL.

Introduction to MySQL

MySQL is a relational database which is used for data storage. You can store any type of data (text, numeric, and alphanumeric). MySQL is used to store your applications data and retrieve it whenever your applications need it. You can update as well as delete the data. So, that's all about the introduction. Now let's see some of the reasons for integrating MySQL with Apache Nutch.

Benefits of integrating MySQL with Apache Nutch

MySQL provides rich querying functionality which other NoSQL stores don't provide. Whichever web pages are crawled by Apache Nutch need to be stored in MySQL. You can use MySQL as a data store with Apache Nutch. As many other data stores are also available, you can use MySQL in the same way. It's a very good data storage option with Apache Nutch. So, that's the benefits of integrating MySQL with Apache Nutch. Now, let's find out about configuration of MySQL with Apache Nutch.

Configuring MySQL with Apache Nutch

In this section, we are going to cover the configuration steps that are required for configuring MySQL with Apache Nutch:

1. Install MySQL Server and MySQL client from the Ubuntu software center, or type the following command:

   ```
   #sudo apt-get install mysql-server mysql-client
   ```

2. Modify `my.cnf` in `/etc/mysql/` for changing MySQL from the default value to Latin using the following command:

   ```
   #sudo vi /etc/mysql/my.cnf
   ```

3. Put the following configuration under `[mysqld]`:

   ```
   innodb_file_format=barracuda
   innodb_file_per_table=true
   innodb_large_prefix=true
   character-set-server=utf8mb4
   collation-server=utf8mb4_unicode_ci
   max_allowed_packet=500M
   ```

4. The `innodb` options help in dealing with the small, primary key size restriction of MySQL. The character and the collation settings are for handling Unicode correctly. The `max_allowed_packet` settings is optional and only necessary for very large sizes. You need to restart your machine for changes to take effect.

5. Check whether MySQL is running by entering the following command:

```
sudo netstat -tap | grep mysql
```

   You should get the output as follows:

```
tcp 0 0 localhost:mysql *:* LISTEN
```

6. We need to set up the Nutch database in MySQL manually because the current Apache Nutch/Apache Gora/MySQL generated database schema is set to Latin. Log in to MySQL by entering the command using the MySQL ID and password which you have set for MySQL:

```
mysql –u <username> -p
```

7. Hit *Enter*. It will ask for a password. Just type your password, and you will be logged in to MySQL.
8. Create the `nutch` database by entering the following command:

```
CREATE DATABASE nutch;
```

9. Hit *Enter* and then to select the `nutch` database, enter the following command:

```
use nutch;
```

10. Hit *Enter* and then create the `webpage` table with the following command:

```
CREATE TABLE 'webpage' (
'id' varchar(767) NOT NULL,
'headers' blob,
'text' longtext DEFAULT NULL,
'status' int(11) DEFAULT NULL,
'markers' blob,
'parseStatus' blob,
'modifiedTime' bigint(20) DEFAULT NULL,
'prevModifiedTime' bigint(20) DEFAULT NULL,
'score' float DEFAULT NULL,
'typ' varchar(32) CHARACTER SET latin1 DEFAULT NULL,
'batchId' varchar(32) CHARACTER SET latin1 DEFAULT NULL,
'baseUrl' varchar(767) DEFAULT NULL,
'content' longblob,
```

```
'title' varchar(2048) DEFAULT NULL,
'reprUrl' varchar(767) DEFAULT NULL,
'fetchInterval' int(11) DEFAULT NULL,
'prevFetchTime' bigint(20) DEFAULT NULL,
'inlinks' mediumblob,
'prevSignature' blob,
'outlinks' mediumblob,
'fetchTime' bigint(20) DEFAULT NULL,
'retriesSinceFetch' int(11) DEFAULT NULL,
'protocolStatus' blob,
'signature' blob,
'metadata' blob,
PRIMARY KEY ('id')
) ENGINE=InnoDB
ROW_FORMAT=COMPRESSED
```

11. Hit *Enter*. You have successfully created the MySQL database for Nutch.

Now, let's understand how to crawl with Apache Nutch on MySQL.

Crawling with Apache Nutch on MySQL

In this section, we are going to integrate Apache Nutch with MySQL, and we will then perform the crawling operation and check whether our crawled web pages come to MySQL database or not. The following are the steps required to do this:

I have used Apache Nutch 2.2.1.

1. Get Apache Nutch 2.2.1 as discussed earlier.
2. Modify `ivy.xml` in `$NUTCH_HOME/ivy/` by changing the following command:

   ```
   <dependency org="org.apache.gora" name="gora-core" rev="0.3″
   conf="*->default"/>
   ```

   to

   ```
   <dependency org="org.apache.gora" name="gora-core" rev="0.2.1″
   conf="*->default"/>
   ```

3. Uncomment `gora-sql` and `mysql-connector-java` as follows:

```
<dependency org="org.apache.gora" name="gora-sql" rev="0.1.1-
incubating" conf="*->default" />
<!- Uncomment this to use MySQL as database with SQL as Gora
store. ->
<dependency org="mysql" name="mysql-connector-java" rev="5.1.18"
conf="*->default"/>
```

4. Modify `gora.properties` in `$NUTCH_HOME/conf/` by deleting or using `#` for commenting out the default `SqlStore` properties. Add the following configuration for MySQL by replacing `xxxxx` with your username and password that you set up for MySQL:

```
################################
# MySQL properties #
################################
gora.sqlstore.jdbc.driver=com.mysql.jdbc.Driver
gora.sqlstore.jdbc.url=jdbc:mysql://localhost:3306/nutch?createDat
abaseIfNotExist=true
gora.sqlstore.jdbc.user=xxxxx
gora.sqlstore.jdbc.password=xxxxx
```

5. Modify the `gora-sql-mapping.xml` file in `$NUTCH_HOME/conf/` by changing the length of the primary key from `512` to `767` as follows. You need to make the change at two places. Just find and change it.

```
<primarykey column="id" length="767"/>
```

6. Modify `nutch-site.xml` in `$NUTCH_HOME/conf/` by adding the following configuration to it:

```
<property>
<name>http.agent.name</name>
<value>YourNutchSpider</value>
</property>
<property>
<name>http.accept.language</name>
<value>ja-jp, en-us,en-gb,en;q=0.7,*;q=0.3</value>
<description>Value of the "Accept-Language" request header field.
This allows selection of non-English language as the default one
to retrieve.
It is a useful setting for search engines built for certain
national groups.
</description>
</property>
<property>
```

```
<name>parser.character.encoding.default</name>
<value>utf-8</value>
<description>The character encoding to fall back to when no other
information
is available</description>
</property>
<property>
<name>storage.data.store.class</name>
<value>org.apache.gora.sql.store.SqlStore</value>
<description>The Gora DataStore class for storing and retrieving
data.
Currently the following stores are available: ….
</description>
</property>
```

7. Install Ant if it's not installed already from the Ubuntu software center or by typing the following command:

   ```
   sudo apt-get install ant
   ```

8. Build Apache Nutch with the following commands:

   ```
   #cd $NUTCH_HOME
   #ant runtime
   ```

9. It will take some time to compile.
10. Create the `seed.txt` file and put URLs in it by entering the following commands:

    ```
    #cd $NUTCH_HOME/runtime/local
    #mkdir -p urls
    #echo 'http://nutch.apache.org/' > urls/seed.txt
    ```

11. Start your Apache Solr, which needs to be integrated with Apache Nutch as discussed in *Chapter 1*, *Getting Started with Apache Nutch*, and then type the following commands for crawling:

    ```
    #Cd $NUTCH_HOME/runtime/local
    # bin/crawl urls/seed.txt TestCrawl http://localhost:8983/solr/ 2
    ```

12. If all succeeds, a table will be created in MySQL with the name `TestCrawl.webpage` as `TestCrawl` is the ID provided in the preceding command and web page is the table name which we have defined earlier.

13. Check your crawled pages' records inside the table by entering the following commands:

```
#mysql -u <username> -p
#use nutch;
#SELECT * FROM TestCrawl.webpage;
```

14. You can also use the MYSQL Query browser to do this. You will get around 320 rows. It will be difficult to read the columns, so you might want to install MySQL Workbench with the following command:

```
sudo apt-get install mysql-workbench
```

15. The following command is used for viewing the data. You might also want to run the following SQL command to limit the number of rows in the `TestCrawl.webpage` table to only select the URLs that were actually parsed:

```
select * from webpage where status = 2;
```

16. You can easily add more URLs to search by hand in `seed.txt` in `$NUTCH_HOME/runtime/local/urls` if you want, and you can then use the following command for injecting URLs:

```
bin/nutch inject urls
```

You have successfully integrated Apache Nutch with MySQL. Now you can use these records and perform whatever operations you require.

Summary

That's the end of this chapter. Let's revise quickly what we have discussed and learned. We started with Apache Accumulo by covering its introduction and other related components—Apache Gora, Apache ZooKeeper, and Apache Hadoop. We saw in detail how we can configure Apache Accumulo with the help of all these mentioned components. We have covered the configuration part of Apache Accumulo and also performed crawling using Apache Nutch on Apache Accumulo. After that, we started with the integration of Apache Nutch with MySQL, where we covered the introduction of MySQL. Then, we covered how MySQL can be integrated with Apache Nutch. We configured MySQL with Apache Nutch successfully, and we also performed crawling using Apache Nutch on MySQL.

Index

A

C

D

E

F

G

H

I

J

L

Thank you for buying

Web Crawling and Data Mining with Apache Nutch

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

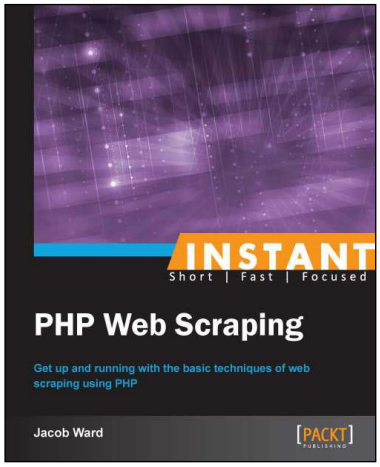
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

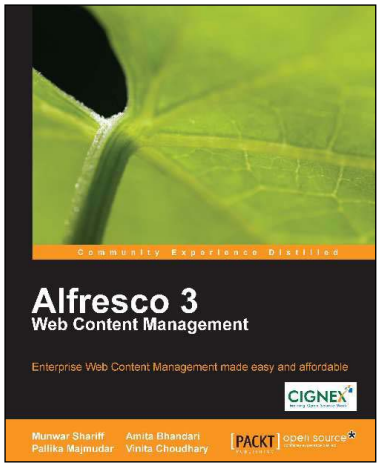


Instant PHP Web Scraping

ISBN: 978-1-78216-476-0 Paperback: 60 pages

Get up and running with the basic techniques of web scraping using PHP

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results
2. Build a re-usable scraping class to expand on for future projects
3. Scrape, parse, and save data from any website with ease



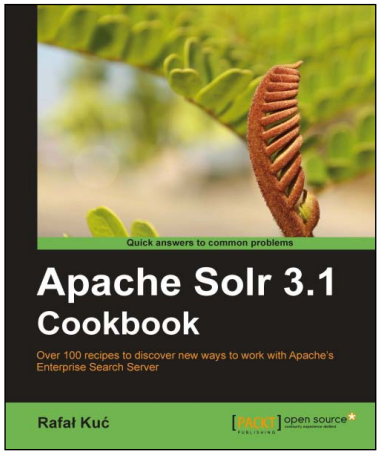
Alfresco 3 Web Content Management

ISBN: 978-1-84719-800-6 Paperback: 440 pages

Enterprise Web Content Management made easy and affordable

1. A complete guide to Web Content Creation and Distribution
2. Understand the concepts and advantages of Publishing-style Web CMS
3. Leverage a single installation to manage multiple websites
4. Integrate Alfresco web applications with external systems

[PACKT] open source
community experience distilled
PUBLISHING



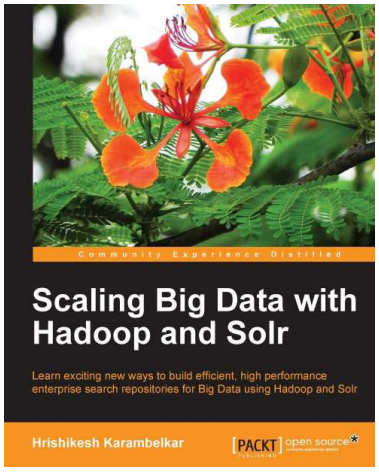
Apache Solr 3.1 Cookbook

ISBN: 978-1-84951-218-3 Paperback: 300 pages

Over 100 recipes to discover new ways to work with Apache's Enterprise Search Server

1. Improve the way in which you work with Apache Solr to make your search engine quicker and more effective
2. Deal with performance, setup, and configuration problems in no time
3. Discover little-known Solr functionalities and create your own modules to customize Solr to your company's needs

---



Scaling Big Data with Hadoop and Solr

ISBN: 978-1-78328-137-4 Paperback: 144 pages

Learn exciting new ways to buid efficient, high performance enterprise search repostories for Big Data using Hadoop and Solr

1. Understand the different approaches of making Solr work on Big Data as well as the benefits and drawbacks
2. Learn from interesting, real-life use cases for Big Data search along with sample code
3. Work with the Distributed Enterprise Search without prior knowledge of Hadoop and Solr