

Jeffrey Aven

Sams **Teach Yourself**
Hadoop

in **24**
Hours

SAMS

800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Hadoop™ in 24 Hours

Copyright © 2017 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33852-6

ISBN-10: 0-672-33852-1

Library of Congress Control Number: 2017935714

Printed in the United States of America

1 17

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearsoned.com.

Editor in Chief

Greg Wiegand

Acquisitions Editor

Trina MacDonald

Development Editor

Chris Zahn

Technical Editor

Adam Shook

Managing Editor

Sandra Schroeder

Project Editor

Lori Lyons

Project Manager

Dhayanidhi

Copy Editor

Abigail Manheim

Indexer

Cheryl Lenser

Proofreader

Sathya Ravi

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Composer

codeMantra

Contents at a Glance

Preface	xii
About the Author	xiv
Acknowledgments	xv

Part I: Getting Started with Hadoop

HOURL 1 Introducing Hadoop	1
2 Understanding the Hadoop Cluster Architecture	11
3 Deploying Hadoop	23
4 Understanding the Hadoop Distributed File System (HDFS)	45
5 Getting Data into Hadoop	63
6 Understanding Data Processing in Hadoop	81

Part II: Using Hadoop

HOURL 7 Programming MapReduce Applications	103
8 Analyzing Data in HDFS Using Apache Pig	125
9 Using Advanced Pig	141
10 Analyzing Data Using Apache Hive	165
11 Using Advanced Hive	185
12 Using SQL-on-Hadoop Solutions	207
13 Introducing Apache Spark	221
14 Using the Hadoop User Environment (HUE)	243
15 Introducing NoSQL	263

Part III: Managing Hadoop

HOURL 16 Managing YARN	279
17 Working with the Hadoop Ecosystem	301
18 Using Cluster Management Utilities	329
19 Scaling Hadoop	355

20	Understanding Cluster Configuration	367
21	Understanding Advanced HDFS	387
22	Securing Hadoop	405
23	Administering, Monitoring, and Troubleshooting Hadoop	421
24	Integrating Hadoop into the Enterprise	443
	Index	453

Table of Contents

Preface	xii
About the Author	xiv
Acknowledgments	xv

Part I: Getting Started with Hadoop

HOOR 1: Introducing Hadoop	1
Hadoop and a Brief History of Big Data	1
Hadoop Explained	2
The Commercial Hadoop Landscape	5
Typical Hadoop Use Cases	6
Summary	8
Q&A	8
Workshop	9
HOOR 2: Understanding the Hadoop Cluster Architecture	11
HDFS Cluster Processes	11
YARN Cluster Processes	14
Hadoop Cluster Architecture and Deployment Modes	17
Summary	20
Q&A	21
Workshop	21
HOOR 3: Deploying Hadoop	23
Installation Platforms and Prerequisites	23
Installing Hadoop	26
Deploying Hadoop in the Cloud	38
Summary	41
Q&A	42
Workshop	42

HOURL 4: Understanding the Hadoop Distributed File System (HDFS)	45
HDFS Overview	45
Review of the HDFS Roles	48
NameNode Metadata	52
SecondaryNameNode Role	55
Interacting with HDFS	56
Summary	59
Q&A	60
Workshop	60
HOURL 5: Getting Data into Hadoop	63
Data Ingestion Using Apache Flume	63
Ingesting Data from a Database using Sqoop	70
Data Ingestion Using HDFS RESTful Interfaces	74
Data Ingestion Considerations	77
Summary	78
Q&A	79
Workshop	79
HOURL 6: Understanding Data Processing in Hadoop	81
Introduction to MapReduce	81
MapReduce Explained	83
Word Count: The “Hello, World” of MapReduce	92
MapReduce in Hadoop	95
Summary	99
Q&A	100
Workshop	100
 Part II: Using Hadoop	
HOURL 7: Programming MapReduce Applications	103
Introducing the Java MapReduce API	103
Writing a MapReduce Program in Java	109

Advanced MapReduce API Concepts	117
Using the MapReduce Streaming API	120
Summary	122
Q&A	123
Workshop	123
HOUR 8: Analyzing Data in HDFS Using Apache Pig	125
Introducing Pig	125
Pig Latin Basics	126
Loading Data into Pig	131
Filtering, Projecting, and Sorting Data using Pig	133
Built-in Functions in Pig	136
Summary	139
Q&A	139
Workshop	140
HOUR 9: Using Advanced Pig	141
Grouping Data in Pig	141
Multiple Dataset Processing in Pig	144
User-Defined Functions in Pig	156
Automating Pig Using Macros and Variables	159
Summary	161
Q&A	162
Workshop	162
HOUR 10: Analyzing Data Using Apache Hive	165
Introducing Hive	165
Creating Hive Objects	171
Analyzing Data with Hive	175
Data Output with Hive	180
Summary	181
Q&A	181
Workshop	182

HOURL 11: Using Advanced Hive	185
Automating Hive	185
Complex Datatypes in Hive	187
Text Processing Using Hive	190
Optimizing and Managing Queries in Hive	199
Summary	204
Q&A	204
Workshop	205
HOURL 12: Using SQL-on-Hadoop Solutions	207
What Is SQL on Hadoop?	207
Columnar Storage in Hadoop	208
Introduction to Impala	211
Introduction to Tez	214
Introduction to HAWQ and Drill	216
Summary	217
Q&A	218
Workshop	218
HOURL 13: Introducing Apache Spark	221
Introducing Spark	221
Spark Architecture	225
Resilient Distributed Datasets in Spark	227
Transformations and Actions in Spark	231
Extensions to Spark	234
Summary	240
Q&A	241
Workshop	241
HOURL 14: Using the Hadoop User Environment (HUE)	243
Introducing HUE	243
Installing, Configuring, and Using HUE	251
Summary	260

Q&A	260
Workshop	261
HOUR 15: Introducing NoSQL	263
Introduction to NoSQL	263
Introducing HBase	265
Introducing Apache Cassandra	273
Other NoSQL Implementations and the Future of NoSQL	275
Summary	276
Q&A	276
Workshop	277
Part III: Managing Hadoop	
HOUR 16: Managing YARN	279
YARN Revisited	279
Administering YARN	285
Application Scheduling in YARN	293
Summary	299
Q&A	299
Workshop	300
HOUR 17: Working with the Hadoop Ecosystem	301
Hadoop Ecosystem Overview	301
Introduction to Oozie	306
Stream Processing and Messaging in Hadoop	311
Infrastructure and Security Projects	315
Machine Learning, Visualization, and More Data Analysis Tools	319
Summary	325
Q&A	326
Workshop	326
HOUR 18: Using Cluster Management Utilities	329
Cluster Management Overview	329
Deploying Clusters and Services Using Management Tools	337

Configuration and Service Management Using Management Tools	340
Monitoring, Troubleshooting, and Securing Hadoop Clusters Using Cluster Management Utilities	347
Getting Started with the Cluster Management Utilities	351
Summary	352
Q&A	353
Workshop	353
HOOR 19: Scaling Hadoop	355
Linear Scalability with Hadoop	355
Adding Nodes to your Hadoop Cluster	356
Decommissioning Nodes from your Cluster	359
Rebalancing a Hadoop Cluster	361
Benchmarking Hadoop	362
Summary	365
Q&A	365
Workshop	366
HOOR 20: Understanding Cluster Configuration	367
Configuration in Hadoop	367
HDFS Configuration Parameters	372
YARN Configuration Parameters	377
Ecosystem Component Configuration	381
Summary	383
Q&A	384
Workshop	385
HOOR 21: Understanding Advanced HDFS	387
HDFS Rack Awareness	387
HDFS High Availability	390
HDFS Federation	398
HDFS Caching, Snapshotting, and Archiving	401
Summary	403
Q&A	403
Workshop	404

HOOR 22: Securing Hadoop	405
Hadoop Security Basics	405
Securing Hadoop with Kerberos	411
Perimeter Security Using Apache Knox	414
Role-Based Access Control Using Ranger and Sentry	416
Summary	418
Q&A	419
Workshop	419
HOOR 23: Administering, Monitoring, and Troubleshooting Hadoop	421
Administering Hadoop	421
Troubleshooting Hadoop	426
System and Application Monitoring in Hadoop	432
Best Practices and Other Information Sources	439
Summary	441
Q&A	441
Workshop	442
HOOR 24: Integrating Hadoop into the Enterprise	443
Hadoop and the Data Center	443
Use Case: Data Warehouse/ETL Offload	445
Use Case: Event Storage and Processing	447
Use Case: Predictive Analytics	448
Summary	450
Q&A	451
Workshop	451
Index	453

Preface

Hadoop is synonymous with Big Data, and the two are inexorably linked together. Although there have been many books written about Hadoop before this one, many of these books have been focused on one particular area of Hadoop, or required some prior experience with Hadoop. This book is different as it explores all areas of Hadoop and the Hadoop ecosystem, as well as providing an understanding and background to the genesis of Hadoop and the big data movement.

This book is also useful if you have had some exposure to Hadoop as it explores adjacent technologies such as Spark, HBase, Cassandra, and more. The book includes many diagrams, code snippets, hands-on exercises, quizzes, and Q&As, which will help you in distilling key concepts.

I have dedicated the past several years of my career to this subject area, teaching courses and consulting to clients on analytics and big data. I have seen the emergence and maturity of the big data and open source movements, and been part of its assimilation into the enterprise. I have tried to synthesize my personal learning journey over this time into this book.

I hope this book launches or assists in your journey to becoming a big data practitioner.

How This Book Is Organized

This book starts by establishing some of the basic concepts behind Hadoop, which are covered in **Part I, “Getting Started with Hadoop.”** I also cover deployment of Hadoop both locally and in the cloud in Part I.

Part II, “Using Hadoop,” is focused on the programming and data interfaces available with Hadoop, which include MapReduce, Pig, Hive, Spark, and more, as well as introductions to SQL-on-Hadoop and NoSQL using HBase.

Part III, “Managing Hadoop,” covers scaling, management, and administration of Hadoop and its various related technologies, including advanced configuration, securing, monitoring, and troubleshooting Hadoop.

Data Used in the Exercises

Data for the Try it Yourself exercises can be downloaded from the book’s Amazon Web Services (AWS) S3 bucket. If you are not familiar with AWS, don’t worry—I cover this topic in the book as well.

Conventions Used in This Book

Each hour begins with “What You’ll Learn in This Hour,” which provides a list of bullet points highlighting the topics covered in that hour. Each hour concludes with a “Summary” page summarizing the main points covered in the hour, as well as “Q&A” and “Quiz” sections to help you consolidate your learning from that hour.

Key topics being introduced for the first time are typically *italicized* by convention. Most hours also include programming examples in numbered code listings. Where functions, commands, classes, or objects are referred to in text, they appear in `monospace` type.

Other asides in this book include the following:

NOTE

Content not integral to the subject matter but worth noting or being aware of.

TIP

TIP Subtitle

A hint or tip relating to the current topic that could be useful.

CAUTION

Caution Subtitle

Something related to the current topic that could lead to issues if not addressed.

TRY IT YOURSELF ▼

Exercise Title

An exercise related to the current topic including a step-by-step guide and descriptions of expected outputs.

About the Author

Jeffrey Aven is a big data, open source software, and cloud computing consultant and instructor based in Melbourne, Australia. Jeff has several years' experience with Hadoop, NoSQL, Spark, and related technologies, and has been involved in key roles with several major big data implementations in Australia. Jeffrey is the author of *SAMS Teach Yourself Apache Spark* and was awarded Cloudera Hadoop Instructor of the Year for APAC in 2013.

Acknowledgments

I would like to acknowledge the team at Pearson who work hard behind the scenes on all of these projects. I would also like to acknowledge Adam Shook for his help as a technical editor for this project.

We Want to Hear from You

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

E-mail: feedback@sampublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

HOUR 3

Deploying Hadoop

What You'll Learn in This Hour:

- ▶ Installation platforms and prerequisites
- ▶ How to install Hadoop from Apache releases
- ▶ How to deploy Hadoop using commercial distributions
- ▶ How to deploy Hadoop in the cloud using AWS

Now that you have been introduced to Hadoop and learned about its core components, HDFS and YARN and their related processes, as well as different deployment modes for Hadoop, let's look at the different options for getting a functioning Hadoop cluster up and running. By the end of this hour you will have set up a working Hadoop cluster that we will use throughout the remainder of the book.

Installation Platforms and Prerequisites

Before you install Hadoop there are a few installation requirements, prerequisites, and recommendations of which you should be aware.

Operating System Requirements

The vast majority of Hadoop implementations are platformed on Linux hosts. This is due to a number of reasons:

- ▶ The Hadoop project, although cross-platform in principle, was originally targeted at Linux. It was several years after the initial release that a Windows-compatible distribution was introduced.
- ▶ Many of the commercial vendors only support Linux.
- ▶ Many other projects in the open source and Hadoop ecosystem have compatibility issues with non-Linux platforms.

That said there are options for installing Hadoop on Windows, should this be your platform of choice. We will use Linux for all of our exercises and examples in this book, but consult the documentation for your preferred Hadoop distribution for Windows installation and support information if required.

If you are using Linux, choose a distribution you are comfortable with. All major distributions are supported (Red Hat, Centos, Ubuntu, SLES, etc.). You can even mix distributions if appropriate; for instance, master nodes running Red Hat and slave nodes running Ubuntu.

CAUTION

Don't Use Logical Volume Manager (LVM) in Linux

If you are using Linux to deploy Hadoop nodes, master or slaves, it is strongly recommended that you not use LVM in Linux. This will restrict performance, especially on slave nodes.

Hardware Requirements

Although there are no hard and fast requirements, there are some general heuristics used in sizing instances, or hosts, appropriately for roles within a Hadoop cluster. First, you need to distinguish between master and slave node instances, and their requirements.

Master Nodes

A Hadoop cluster relies on its master nodes, which host the NameNode and ResourceManager, to operate, although you can implement high availability for each subsystem as I discussed in the last hour. Failure and failover of these components is not desired. Furthermore, the master node processes, particularly the NameNode, require a large amount of memory to operate efficiently, as you will appreciate in the next hour when we dive into the internals of HDFS. Therefore, when specifying hardware requirements the following guidelines can be used for medium to large-scale production Hadoop implementations:

- ▶ 16 or more CPU cores (preferably 32)
- ▶ 128GB or more RAM (preferably 256GB)
- ▶ RAID Hard Drive Configuration (preferably with hot-swappable drives)
- ▶ Redundant power supplies
- ▶ Bonded Gigabit Ethernet or 10Gigabit Ethernet

This is only a guide, and as technology moves on quickly, these recommendations will change as well. The bottom line is that you need *carrier class* hardware with as much CPU and memory capacity as you can get!

Slave Nodes

Slave nodes do the actual work in Hadoop, both for processing and storage so they will benefit from more CPU and memory—physical memory, not virtual memory. That said, slave nodes are designed with the expectation of failure, which is one of the reasons blocks are replicated in HDFS. Slave nodes can also be scaled out linearly. For instance, you can simply add more nodes to add more aggregate storage or processing capacity to the cluster, which you cannot do with master nodes. With this in mind, economic scalability is the objective when it comes to slave nodes. The following is a guide for slave nodes for a well-subscribed, computationally intensive Hadoop cluster; for instance, a cluster hosting machine learning and in memory processing using Spark.

- ▶ 16-32 CPU cores
- ▶ 64-512 GB of RAM
- ▶ 12-24 1-4 TB hard disks in a JBOD Configuration

NOTE

JBOD

JBOD is an acronym for *just a bunch of disks*, meaning directly attached storage that is not in a RAID configuration, where each disk operates independently of the other disks. RAID is not recommended for block storage on slave nodes as the access speed is limited by the slowest disk in the array, unlike JBOD where the average speed can be greater than that of the slowest disk. JBOD has been proven to outperform RAID 0 for block storage by 30% to 50% in benchmarks conducted at Yahoo!.

CAUTION

Storing Too Much Data on Any Slave Node May Cause Issues

As slave nodes typically host the blocks in a Hadoop filesystem, and as storage costs, particularly for JBOD configurations, are relatively inexpensive, it may be tempting to allocate excess block storage capacity to each slave node. However, as you will learn in the next hour on HDFS, you need to consider the network impact of a failed node, which will trigger re-replication of all blocks that were stored on the slave node.

Slave nodes are designed to be deployed on commodity-class hardware, and yet while they still need ample processing power in the form of CPU cores and memory, as they will be executing computational and data transformation tasks, they don't require the same degree of fault tolerance that master nodes do.

Networking Considerations

Fully distributed Hadoop clusters are very chatty, with control messages, status updates and heartbeats, block reports, data shuffling, and block replication, and there is often heavy network utilization between nodes of the cluster. If you are deploying Hadoop on-premise, you should always deploy Hadoop clusters on private subnets with dedicated switches. If you are using multiple racks for your Hadoop cluster (you will learn more about this in **Hour 21**, “**Understanding Advanced HDFS**”), you should consider redundant core and “top of rack” switches.

Hostname resolution is essential between nodes of a Hadoop cluster, so both forward and reverse DNS lookups must work correctly between each node (master-slave and slave-slave) for Hadoop to function. Either DNS or a `hosts` files can be used for resolution. IPv6 should also be disabled on all hosts in the cluster.

Time synchronization between nodes of the cluster is essential as well, as some components, such as Kerberos, which is discussed in **Hour 22**, “**Securing Hadoop**,” rely on this being the case. It is recommended you use `ntp` (Network Time Protocol) to keep clocks synchronized between all nodes.

Software Requirements

As discussed, Hadoop is almost entirely written in Java and compiled to run in a Java Runtime Environment (JRE); therefore Java is a prerequisite to installing Hadoop. Current prerequisites include:

- ▶ Java Runtime Environment (JRE) 1.7 or above
- ▶ Java Development Kit (JDK) 1.7 or above—required if you will be compiling Java classes such as MapReduce applications

Other ecosystem projects will have their specific prerequisites; for instance, Apache Spark requires Scala and Python as well, so you should always refer to the documentation for these specific projects.

Installing Hadoop

You have numerous options for installing Hadoop and setting up Hadoop clusters. As Hadoop is a top-level Apache Software Foundation (ASF) open source project, one method is to install directly from the Apache builds on <http://hadoop.apache.org/>. To do this you first need one or more hosts, depending upon the mode you wish to use, with appropriate hardware specifications, an appropriate operating system, and a Java runtime environment available (all of the prerequisites and considerations discussed in the previous section).

Once you have this, it is simply a matter of downloading and unpacking the desired release. There may be some additional configuration to be done afterwards, but then you simply start the relevant services (master and slave node daemons) on their designated hosts and you are up and running.

Non-Commercial Hadoop

Let's deploy a Hadoop cluster using the latest Apache release now.

TRY IT YOURSELF ▼

Installing Hadoop Using the Apache Release

In this exercise we will install a pseudo-distributed mode Hadoop cluster using the latest Hadoop release downloaded from hadoop.apache.org.

As this is a test cluster the following specifications will be used in our example:

- ▶ Red Hat Enterprise Linux 7.2 (The installation steps would be similar using other Linux distributions such as Ubuntu)
- ▶ 2 CPU cores
- ▶ 8GB RAM
- ▶ 30GB HDD
- ▶ hostname: **hadoopnode0**

1. Disable SELinux (this is known to cause issues with Hadoop):

```
$ sudo sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' \
/etc/selinux/config
```

2. Disable IPv6 (this is also known to cause issues with Hadoop):

```
$ sudo sed -i "\$anet.ipv6.conf.all.disable_ipv6 = 1" \
/etc/sysctl.conf
$ sudo sed -i "\$anet.ipv6.conf.default.disable_ipv6 = 1" \
/etc/sysctl.conf
$ sudo sysctl -p
```

3. Reboot

4. Run the `sestatus` command to ensure SELinux is not enabled:

```
$ sestatus
```

- 5.** Install Java. We will install the OpenJDK, which will install both a JDK and JRE:

```
$ sudo yum install java-1.7.0-openjdk-devel
```

- a.** Test that Java has been successfully installed by running the following command:

```
$ java -version
```

If Java has been installed correctly you should see output similar to the following:

```
java version "1.7.0_101"
OpenJDK Runtime Environment (rhel-2.6.6.1.el7_2-x86_64..)
OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode)
```

Note that depending upon which operating system you are deploying on, you may have a version of Java and a JDK installed already. In these cases it may not be necessary to install the JDK, or you may need to set up alternatives so you do not have conflicting Java versions.

- 6.** Locate the installation path for Java, and set the `JAVA_HOME` environment variable:

```
$ export JAVA_HOME=/usr/lib/jvm/REPLACE_WITH_YOUR_PATH/
```

- 7.** Download Hadoop from your nearest Apache download mirror. You can obtain the link by selecting the binary option for the version of your choice at <http://hadoop.apache.org/releases.html>. We will use Hadoop version 2.7.2 for our example.

```
$ wget http://REPLACE_WITH_YOUR_MIRROR/hadoop-2.7.2.tar.gz
```

- 8.** Unpack the Hadoop release, move it into a system directory, and set an environment variable from the Hadoop home directory:

```
$ tar -xvf hadoop-2.7.2.tar.gz
$ mv hadoop-2.7.2 hadoop
$ sudo mv hadoop/ /usr/share/
$ export HADOOP_HOME=/usr/share/hadoop
```

- 9.** Create a directory which we will use as an alternative to the Hadoop configuration directory:

```
$ sudo mkdir -p /etc/hadoop/conf
```

- 10.** Create a `mapred-site.xml` file (I will discuss this later) in the Hadoop configuration directory:

```
$ sudo cp $HADOOP_HOME/etc/hadoop/mapred-site.xml.template \
$HADOOP_HOME/etc/hadoop/mapred-site.xml
```

11. Add `JAVA_HOME` environment variable to `hadoop-env.sh` (file used to source environment variables for Hadoop processes):

```
$ sed -i "\$aexport JAVA_HOME=/REPLACE_WITH_YOUR_JDK_PATH/" \  
$HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

Substitute the correct path to your Java home directory as defined in Step 6.

12. Create a symbolic link between the Hadoop configuration directory and the `/etc/hadoop/conf` directory created in Step 10:

```
$ sudo ln -s $HADOOP_HOME/etc/hadoop/* \  
/etc/hadoop/conf/
```

13. Create a logs directory for Hadoop:

```
$ mkdir $HADOOP_HOME/logs
```

14. Create users and groups for HDFS and YARN:

```
$ sudo groupadd hadoop  
$ sudo useradd -g hadoop hdfs  
$ sudo useradd -g hadoop yarn
```

15. Change the group and permissions for the Hadoop release files:

```
$ sudo chgrp -R hadoop /usr/share/hadoop  
$ sudo chmod -R 777 /usr/share/hadoop
```

16. Run the built in Pi Estimator example included with the Hadoop release.

```
$ cd $HADOOP_HOME  
$ sudo -u hdfs bin/hadoop jar \  
share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar \  
pi 16 1000
```

As we have not started any daemons or initialized HDFS, this program runs in `LocalJobRunner` mode (recall that I discussed this in **Hour 2, “Understanding the Hadoop Cluster Architecture”**). If this runs correctly you should see output similar to the following:

```
...  
Job Finished in 2.571 seconds  
Estimated value of Pi is 3.14250000000000000000
```

Now let's configure a pseudo-distributed mode Hadoop cluster from your installation.

17. Use the `vi` editor to update the `core-site.xml` file, which contains important information about the cluster, specifically the location of the namenode:

```
$ sudo vi /etc/hadoop/conf/core-site.xml
# add the following config between the <configuration>
# and </configuration> tags:
<property>
<name>fs.defaultFS</name>
<value>hdfs://hadoopnode0:8020</value>
</property>
```

Note that the value for the `fs.defaultFS` configuration parameter needs to be set to `hdfs://HOSTNAME:8020`, where the `HOSTNAME` is the name of the NameNode host, which happens to be the `localhost` in this case.

18. Adapt the instructions in Step 17 to similarly update the `hdfs-site.xml` file, which contains information specific to HDFS, including the replication factor, which is set to 1 in this case as it is a pseudo-distributed mode cluster:

```
sudo vi /etc/hadoop/conf/hdfs-site.xml
# add the following config between the <configuration>
# and </configuration> tags:
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
```

19. Adapt the instructions in Step 17 to similarly update the `yarn-site.xml` file, which contains information specific to YARN. Importantly, this configuration file contains the address of the resource manager for the cluster—in this case it happens to be the `localhost`, as we are using pseudo-distributed mode:

```
$ sudo vi /etc/hadoop/conf/yarn-site.xml
# add the following config between the <configuration>
# and </configuration> tags:
<property>
<name>yarn.resourcemanager.hostname</name>
<value>hadoopnode0</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
```

20. Adapt the instructions in Step 17 to similarly update the `mapred-site.xml` file, which contains information specific to running MapReduce applications using YARN:

```
$ sudo vi /etc/hadoop/conf/mapred-site.xml
# add the following config between the <configuration>
```



```
# and </configuration> tags:
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
```

21. Format HDFS on the NameNode:

```
$ sudo -u hdfs bin/hdfs namenode -format
```

Enter [Y] to re-format if prompted.

22. Start the NameNode and DataNode (HDFS) daemons:

```
$ sudo -u hdfs sbin/hadoop-daemon.sh start namenode
$ sudo -u hdfs sbin/hadoop-daemon.sh start datanode
```

23. Start the ResourceManager and NodeManager (YARN) daemons:

```
$ sudo -u yarn sbin/yarn-daemon.sh start resourcemanager
$ sudo -u yarn sbin/yarn-daemon.sh start nodemanager
```

24. Use the `jps` command included with the Java JDK to see the Java processes that are running:

```
$ sudo jps
```

You should see output similar to the following:

```
2374 DataNode
2835 Jps
2280 NameNode
2485 ResourceManager
2737 NodeManager
```

25. Create user directories and a `tmp` directory in HDFS and set the appropriate permissions and ownership:

```
$ sudo -u hdfs bin/hadoop fs -mkdir -p /user/<your_user>
$ sudo -u hdfs bin/hadoop fs -chown <your_user>:<your_user> /user/<your_user>
$ sudo -u hdfs bin/hadoop fs -mkdir /tmp
$ sudo -u hdfs bin/hadoop fs -chmod 777 /tmp
```

26. Now run the same Pi Estimator example you ran in Step 16. This will now run in pseudo-distributed mode:

```
$ bin/hadoop jar \
share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar \
pi 16 1000
```

The output you will see in the console will be similar to that in Step 16. Open a browser and go to localhost:8088. You will see the YARN ResourceManager Web UI (which I discuss in **Hour 6, “Understanding Data Processing in Hadoop”**) (**Figure 3.1**):

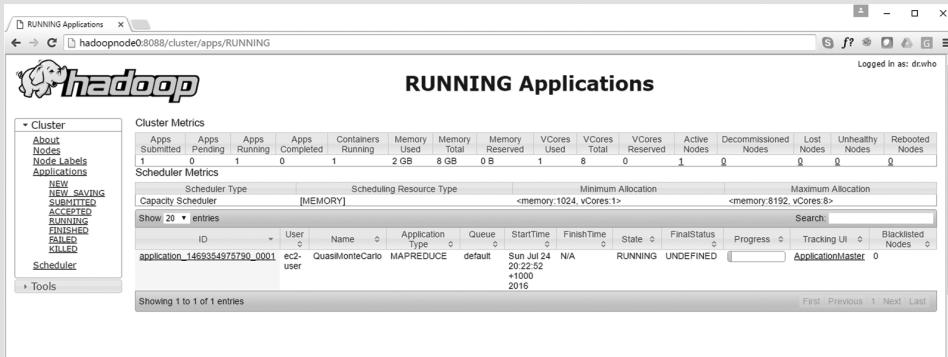


FIGURE 3.1
YARN ResourceManager Web UI.

Congratulations! You have just set up your first pseudo-distributed mode Hadoop cluster. We will use this cluster in other exercises in the book, so keep it available if you can.

Using a Commercial Hadoop Distribution

As I had discussed in **Hour 1, “Introducing Hadoop,”** the commercial Hadoop landscape is well established. With the advent of the ODPI (the Open Data Platform initiative), a once-numerous array of vendors and derivative distributions has been consolidated to a much simpler landscape which includes three primary *pure-play* Hadoop vendors:

- ▶ Cloudera
- ▶ Hortonworks
- ▶ MapR

Importantly, enterprise support agreements and subscriptions can be purchased from the various Hadoop vendors for their distributions. Each vendor also supplies a suite of management utilities to help you deploy and manage Hadoop clusters. Let’s look at each of the three major pure play Hadoop vendors and their respective distributions.

Cloudera

Cloudera was the first mover in the commercial Hadoop space, establishing their first commercial release in 2008. Cloudera provides a Hadoop distribution called CDH (Cloudera Distribution of Hadoop), which includes the Hadoop core and many ecosystem projects. CDH is entirely open source.

Cloudera also provides a management utility called Cloudera Manager (which is not open source). Cloudera Manager provides a management console and framework enabling the deployment, management, and monitoring of Hadoop clusters, and which makes many administrative tasks such as setting up high availability or security much easier. The mix of open source and proprietary software is often referred to as **open core**. A screenshot showing Cloudera Manager is pictured in Figure 3.2.

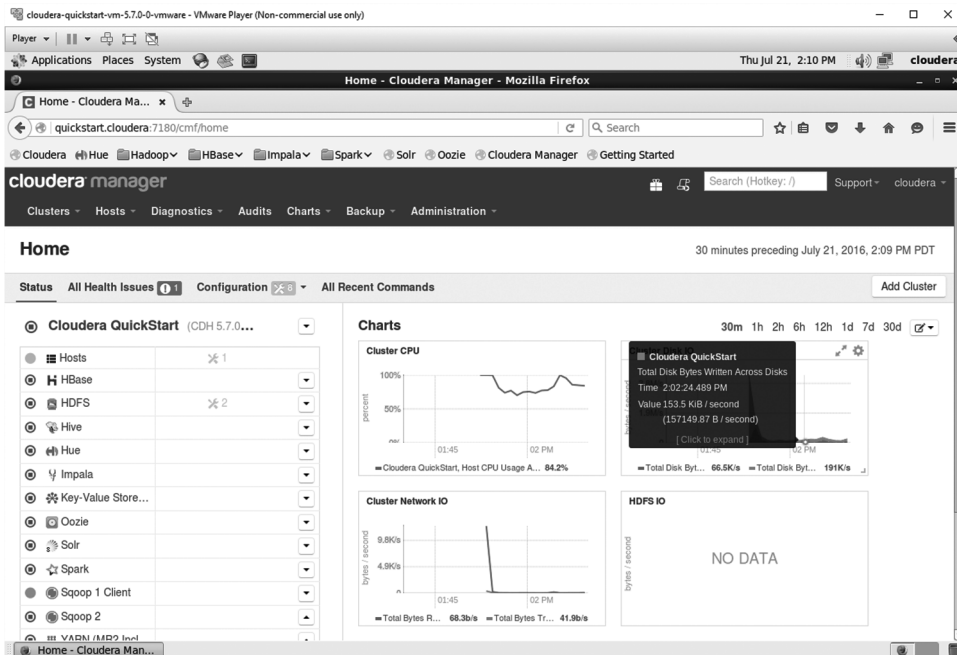


FIGURE 3.2
Cloudera Manager.

As mentioned, Cloudera Manager can be used to deploy Hadoop clusters, including master nodes, slave nodes, and ecosystem technologies. Cloudera Manager distributes installation packages for Hadoop components through a mechanism called *parcels*. As Hadoop installations are typically isolated from public networks, Cloudera Manager, which is technically not part of the cluster and will often have access to the Internet, will download parcels and distribute

these to new target hosts nominated to perform roles in a Hadoop cluster or to existing hosts to upgrade components.

Deploying a fully distributed CDH cluster using Cloudera Manager would involve the following steps at a high level:

1. Install Cloudera Manager on a host that has access to other hosts targeted for roles in the cluster.
2. Specify target hosts for the cluster using Cloudera Manager.
3. Use Cloudera Manager to provision Hadoop services, including master and slave nodes.

Cloudera also provides a Quickstart virtual machine, which is a pre-configured pseudo-distributed Hadoop cluster with the entire CDH stack, including core and ecosystem components, as well as a Cloudera Manager installation. This virtual machine is available for VirtualBox, VMware, and KVM, and works with the free editions of each of the virtualization platforms. The Cloudera Quickstart VM is pictured in Figure 3.3.

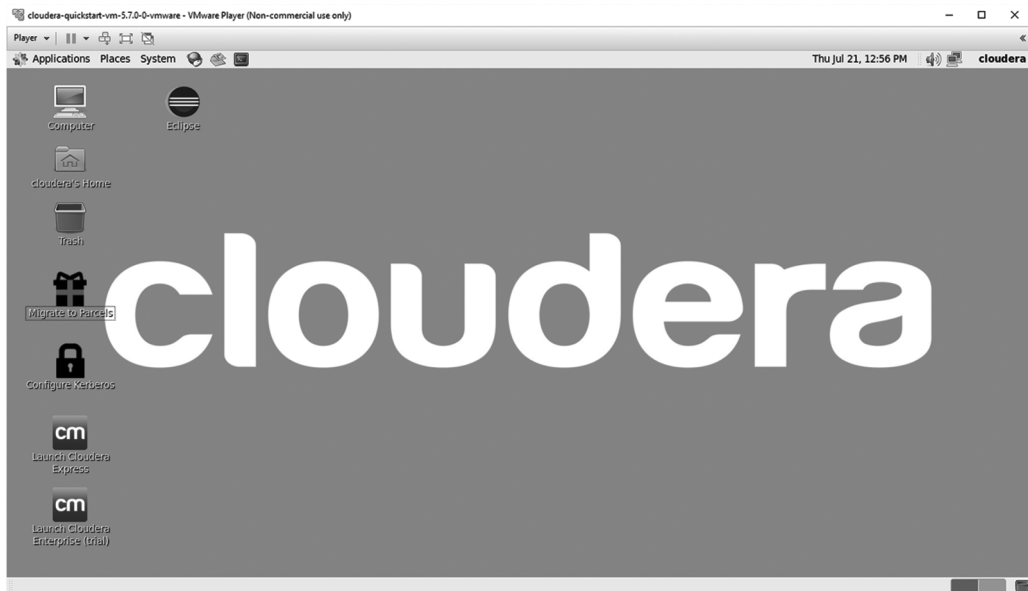


FIGURE 3.3
Cloudera Quickstart VM.

The Quickstart VM is a great way to get started with the Cloudera Hadoop offering. To find out more, go to <http://www.cloudera.com/downloads.html>.

More information about Cloudera is available at <http://www.cloudera.com/>.

Hortonworks

Hortonworks provides pure open source Hadoop distribution and a founding member of the open data platform initiative (ODPi) discussed in **Hour 1**. Hortonworks delivers a distribution called HDP (Hortonworks Data Platform), which is a complete Hadoop stack including the Hadoop core and selected ecosystem components. Hortonworks uses the Apache Ambari project to provide its deployment configuration management and cluster monitoring facilities. A screenshot of Ambari is shown in Figure 3.4.

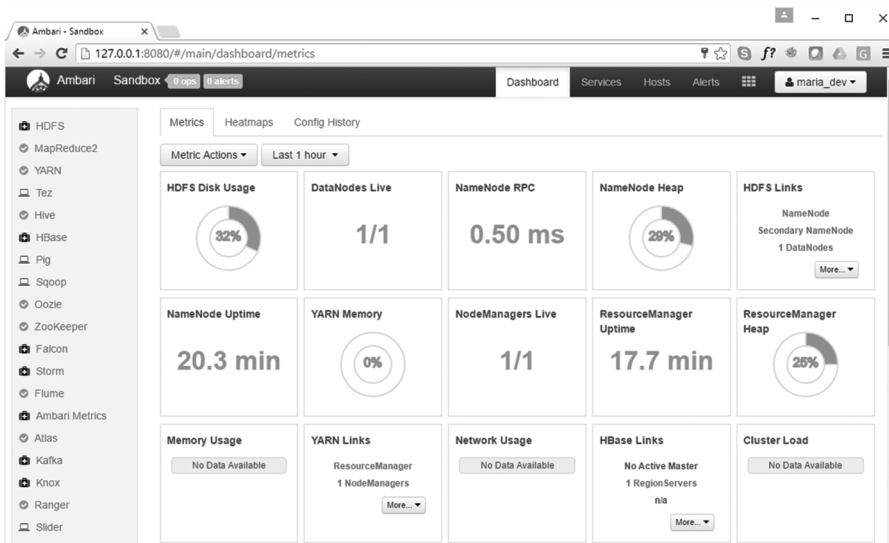


FIGURE 3.4
Ambari console.

The simplest method to deploy a Hortonworks Hadoop cluster would involve the following steps:

1. Install Ambari using the Hortonworks installer on a selected host.
2. Add hosts to the cluster using Ambari.
3. Deploy Hadoop services (such as HDFS and YARN) using Ambari.

Hortonworks provides a fully functional, pseudo-distributed HDP cluster with the complete Hortonworks application stack in a virtual machine called the Hortonworks Sandbox. The Hortonworks Sandbox is available for VirtualBox, VMware, and KVM. The Sandbox virtual machine includes several demo applications and learning tools to use to explore Hadoop and its various projects and components. The Hortonworks Sandbox welcome screen is shown in Figure 3.5.

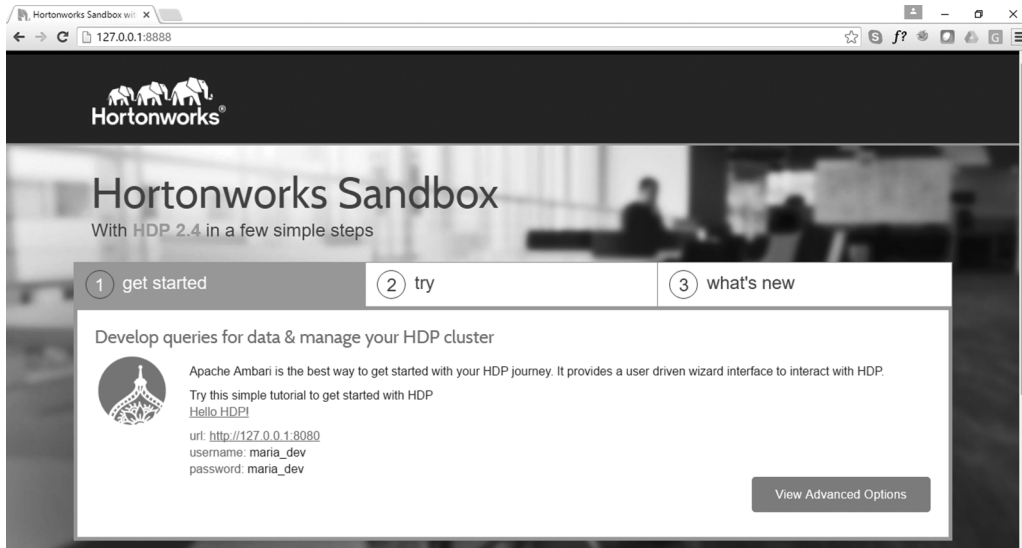


FIGURE 3.5
Hortonworks Sandbox.

You can download the Hortonworks Sandbox from <http://hortonworks.com/products/sandbox/>. More information about Hortonworks and HDP can be obtained from <http://hortonworks.com/>.

MapR

MapR delivers a “Hadoop-derived” software platform that implements an API-compatible distributed filesystem called MapRFS (the MapR distributed Filesystem). MapRFS has been designed to maximize performance and provide read-write capabilities not offered by native HDFS. MapR delivers three versions of their offering called the “Converged Data Platform.” These include:

- ▶ M3 or “Converged Community Edition” (free version)
- ▶ M5 or “Converged Enterprise Edition” (supported version)
- ▶ M7 (M5 version that includes MapR’s custom HBase-derived data store)

Like the other distributions, MapR has a demo offering called the “MapR Sandbox,” which is available for VirtualBox or VMware. It is pictured in Figure 3.6.

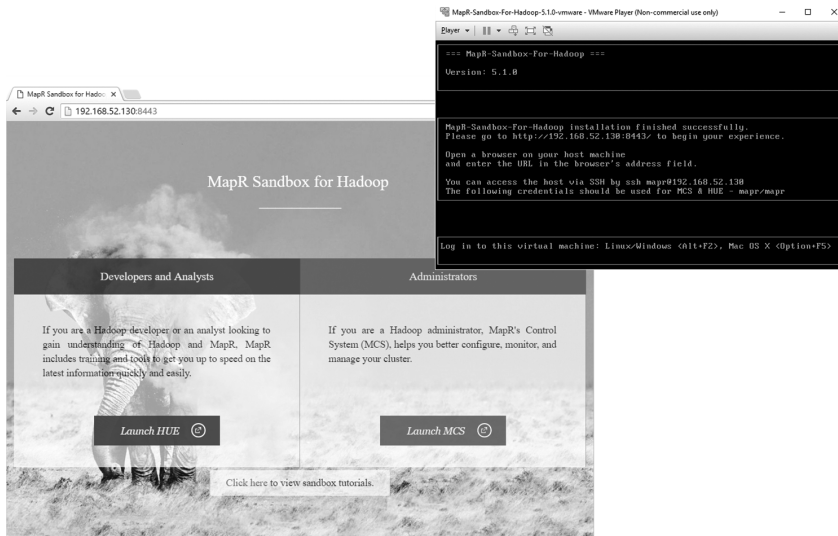


FIGURE 3.6
MapR Sandbox VM.

The MapR Sandbox can be downloaded from <https://www.mapr.com/products/mapr-sandbox-hadoop/download>.

MapR's management offering is called the MapR Control System (MCS), which offers a central console to configure, monitor and manage MapR clusters. It is shown in Figure 3.7.

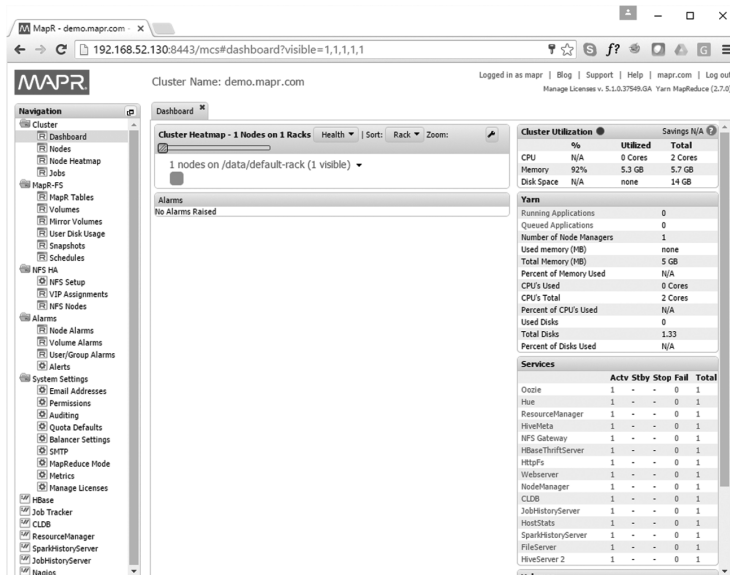


FIGURE 3.7
MapR Control System (MCS).

Much more information about MapR and the Converged Data Platform is available at <https://www.mapr.com/>.

Deploying Hadoop in the Cloud

The rise and proliferation of cloud computing and virtualization technologies has definitely been a game changer for the way organizations think about and deploy technology, and Hadoop is no exception. The availability and maturity around IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service) and SaaS (Software-as-a-Service) solutions makes deploying Hadoop in the cloud not only viable but, in some cases, desirable.

There are many public cloud variants that can be used to deploy Hadoop including Google, IBM, Rackspace, and others. Perhaps the most pervasive cloud platform to date has been AWS (Amazon Web Services), which I will use as the basis for our discussions.

Before you learn about deployment options for Hadoop in AWS, let's go through a quick primer and background on some of the key AWS components. If you are familiar with AWS, feel free to jump straight to the Try it Yourself exercise on deploying Hadoop using AWS EMR.

EC2

Elastic Compute Cloud (EC2) EC2 is Amazon's web service-enabled virtual computing platform. EC2 enables users to create virtual servers and networks in the cloud. The virtual servers are called instances. EC2 instances can be created with a variety of different instance permutations. The Instance Type property determines the number of virtual CPUs and the amount of memory and storage an EC2 instance has available to it. An example instance type is m4.large. A complete list of the different EC2 instance types available can be found at <https://aws.amazon.com/ec2/instance-types/>.

EC2 instances can be optimized for compute, memory, storage and mixed purposes and can even include GPUs (Graphics Processing Units), a popular option for machine learning and deep analytics.

There are numerous options for operating systems with EC2 instances. All of the popular Linux distributions are supported, including Red Hat, Ubuntu, and SLES, as well various Microsoft Windows options.

EC2 instances are created in security groups. Security groups govern network permissions and Access Control Lists (ACLs). Instances can also be created in a Virtual Private Cloud (VPC). A VPC is a private network, not exposed directly to the Internet. This is a popular option for organizations looking to minimize exposure of EC2 instances to the public Internet.

EC2 instances can be provisioned with various storage options, including instance storage or ephemeral storage, which are terms for volatile storage that is lost when an instance is stopped,

and Elastic Block Store (EBS), which is persistent, fault-tolerant storage. There are different options with each, such as SSD (solid state) for instance storage, or provisioned IOPS with EBS.

Additionally, AWS offers Spot instances, which enable you to bid on spare Amazon EC2 computing capacity, often available at a discount compared to normal on-demand EC2 instance pricing.

EC2, as well as all other AWS services, is located in an AWS region. There are currently nine regions, which include the following:

- ▶ US East (N. Virginia)
- ▶ US West (Oregon)
- ▶ US West (N. California)
- ▶ EU (Ireland)
- ▶ EU (Frankfurt)
- ▶ Asia Pacific (Singapore)
- ▶ Asia Pacific (Sydney)
- ▶ Asia Pacific (Tokyo)
- ▶ South America (Sao Paulo)

S3

Simple Storage Service (S3) is Amazon's cloud-based object store. An object store manages data (such as files) as objects. These objects exist in buckets. Buckets are logical, user-created containers with properties and permissions. S3 provides APIs for users to create and manage buckets as well as to create, read, and delete objects from buckets.

The S3 bucket namespace is global, meaning that any buckets created must have a globally unique name. The AWS console or APIs will let you know if you are trying to create a bucket with a name that already exists.

S3 objects, like files in HDFS, are immutable, meaning they are write once, read many. When an S3 object is created and uploaded, an ETag is created, which is effectively a signature for the object. This can be used to ensure integrity when the object is accessed (downloaded) in the future.

There are also public buckets in S3 containing public data sets. These are datasets provided for informational or educational purposes, but they can be used for data operations such as processing with Hadoop. Public datasets, many of which are in the tens or hundreds of terabytes, are available, and topics range from historical weather data to census data, and from astronomical data to genetic data.

More information about S3 is available at <https://aws.amazon.com/s3/>.

Elastic MapReduce (EMR)

Elastic MapReduce (EMR) is Amazon’s Hadoop-as-a-Service platform. EMR clusters can be provisioned using the AWS Management Console or via the AWS APIs. Options for creating EMR clusters include number of nodes, node instance types, Hadoop distribution, and additional ecosystem applications to install.

EMR clusters can read data and output results directly to and from S3. They are intended to be provisioned on demand, run a discrete workflow, a job flow, and terminate. They do have local storage, but they are not intended to run in perpetuity. You should only use this local storage for transient data.

EMR is a quick and scalable deployment method for Hadoop. More information about EMR can be found at <https://aws.amazon.com/elasticmapreduce/>.

AWS Pricing and Getting Started

AWS products, including EC2, S3, and EMR, are charged based upon usage. Each EC2 instance type within each region has an instance per hour cost associated with it. The usage costs per hour are usually relatively low and the medium- to long-term costs are quite reasonable, but the more resources you use for a longer period of time, the more you are charged.

If you have not already signed up with AWS, you’re in luck! AWS has a free tier available for new accounts that enables you to use certain instance types and services for free for the first year. You can find out more at <https://aws.amazon.com/free/>. This page walks you through setting up an account with no ongoing obligations.

Once you are up and running with AWS, you can create an EMR cluster by navigating to the EMR link in the AWS console as shown in Figure 3.8.

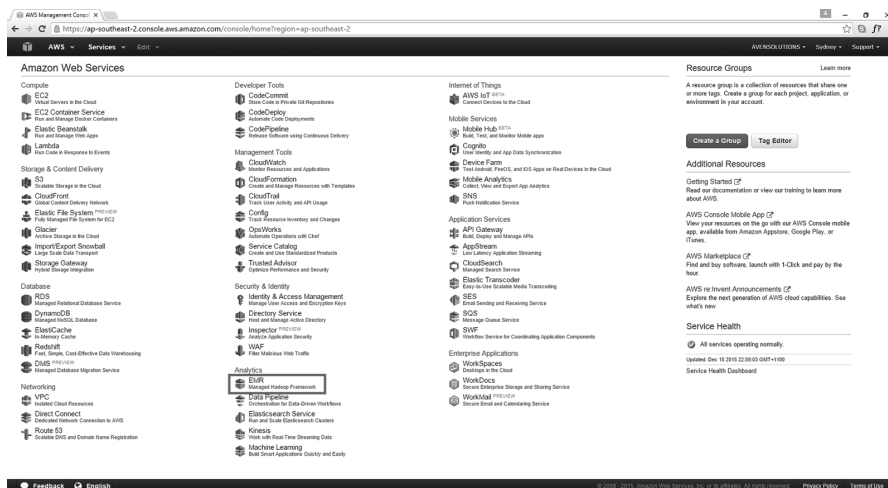


FIGURE 3.8
AWS console—EMR option.

Then click **Create Cluster** on the EMR welcome page as shown in Figure 3.9, and simply follow the dialog prompts.

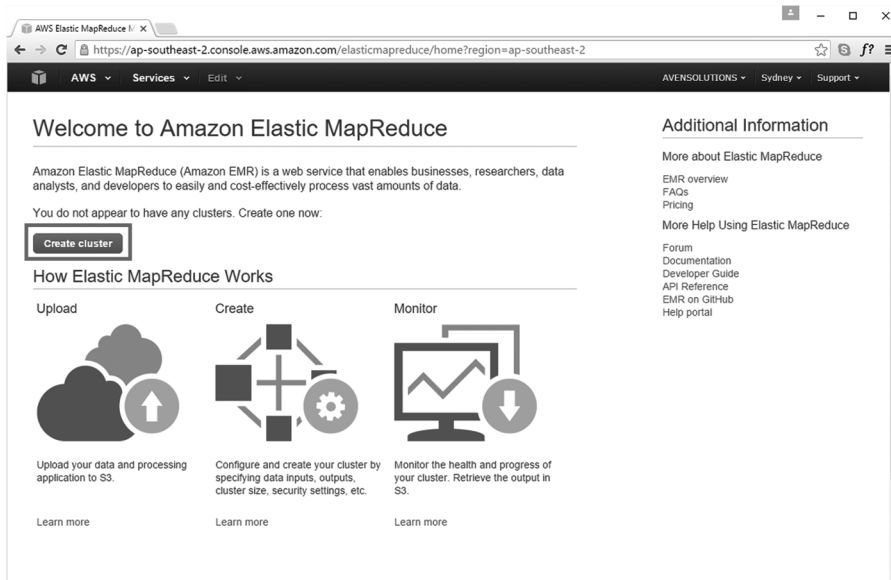


FIGURE 3.9
AWS EMR welcome screen.

You can use an EMR cluster for many of our exercises. However, be aware that leaving the cluster up and running will incur usage charges.

Summary

In this hour you learned about the various approaches to deploying a Hadoop cluster including the Apache releases, commercial distributions and cloud deployment options. Commercial distributions are often the best approach to deploying Hadoop on premise in most organizations as these distributions provide a stable, tested combination of core and ecosystem releases, as well as typically providing a suite of management capabilities useful for deploying and managing Hadoop clusters at scale.

You also learned how to provision Hadoop clusters in the cloud by using the Amazon Web Services Hadoop-as-a-Service offering—Elastic MapReduce (EMR). You are encouraged to explore all the options available to deploy Hadoop. As you progress through the book you will be performing hands-on exercises using Hadoop, so you will need to have a functional cluster. This could be one of the sandbox or quickstart commercial offerings or the Apache Hadoop cluster we set up in the Try it Yourself exercise in this hour.

Q&A

Q. Why do master nodes normally require a higher degree of fault tolerance than slave nodes?

A. Slave nodes are designed to be implemented on commodity hardware with the expectation of failure, and this enables slave nodes to scale economically. The fault tolerance and resiliency built into HDFS and YARN enables the system to recover seamlessly from a failed slave node. Master nodes are different; they are intended to be “always on.” Although there are high availability implementation options for master nodes, failover is not desirable. Therefore, more local fault tolerance, such as RAID disks, dual power supplies, etc., is preferred for master nodes.

Q. What does JBOD stand for, and what is its relevance for Hadoop?

A. JBOD is an acronym for “Just a Bunch of Disks,” which means spinning disks that operate independently of one another, in contrast to RAID, where disks operate as an array. JBOD is recommended for slave nodes, which are responsible for HDFS block storage. This is because the average speed of all disks on a slave node is greater than the speed of the slowest disk. By comparison, RAID read and write speeds are limited by the speed of the slowest disk in the array.

Q. What are the advantages to deploying Hadoop using a commercial distribution?

A. Commercial distributions contain a “stack” of core and ecosystem components that are tested with one another and certified for the respective distribution. The commercial vendors typically include a management application, which is very useful for managing multi-node Hadoop clusters at scale. The commercial vendors also offer enterprise support as an option as well.

Workshop

The workshop contains quiz questions to help you solidify your understanding of the material covered. Try to answer all questions before looking at the “Answers” section that follows.

Quiz

- 1. True or False:** A Java Runtime Environment (JRE) is required on hosts that run Hadoop services.
- 2. Which AWS PaaS product is used to deploy Hadoop as a service?**
 - A.** EC2
 - B.** EMR
 - C.** S3
 - D.** DynamoDB

3. Slave nodes are typically deployed on what class of hardware?
4. The open source Hadoop cluster management utility used by Hortonworks is called _____.

Answers

1. True. Hadoop services and processes are written in Java, are compiled to Java bytecode, and run in Java Virtual Machines (JVMs), and therefore require a JRE to operate.
2. B. Elastic MapReduce (EMR).
3. Commodity.
4. Ambari.

Index

A

- access control policies, 416–418
- accumulators, 436
- Accumulo, 305–306
- ACLs (access control lists) in HDFS, 53–54
- action nodes, 307
- actions on RDDs, 231
- ADD PARTITION statement (Hive), 200
- administration
 - of HDFS, 422–424
 - of Hive, 425–426
 - of YARN, 424–425
- advanced analytics use case, 7–8
- aged history offload, 445
- agents (Flume), 64–65
 - channels, 67
 - sinks, 66–67
 - sources, 65–66
- aggregating data
 - in Hive, 176
 - in Pig, 143
- Amazon Web Services (AWS)
 - deploying Hadoop clusters, 38–41
 - EC2, 38–39
 - EMR, 40
 - pricing, 40–41
 - S3, 39
- Ambari. *See* Apache Ambari
- Ambari Views, 335–336
- analyzing data in Hive, 175
- Apache Ambari, 333–336
 - Ambari Views, 335–336
 - architecture, 334–335
 - configuration management, 345–346
 - deploying services, 337–340
 - installing, 335, 351
 - monitoring and troubleshooting clusters, 347–350
 - securing clusters, 351
 - service management, 344–345
- Apache Avro project, 105
- Apache Cassandra. *See* Cassandra
- Apache DataFu, 158

Apache Drill, 208, 216–217
Apache Flink, 208
Apache Flume, 63–70
 architecture, 64–67
 installing, 67–70
Apache HAWQ, 208, 216
Apache Hive. *See* **Hive**
Apache Kafka, 313–315, 447
Apache Knox, 415–416
Apache Mahout, 320, 450
Apache Parquet, 209–210
Apache Pig. *See* **Pig**
Apache Presto, 208
Apache Ranger, 417
Apache Sentry, 418
Apache Software Foundation (ASF), 2
Apache Spark. *See* **Spark**
Apache Storm, 311–312
Apache Tez, 208, 214–216
Apache Thrift, 319
Apache Zeppelin, 323–324
appenders (Log4j), 427
ApplicationMaster, 16, 96, 281, 283
applications
 log files within, 430–431
 MapReduce
 compiling and packaging, 114–116
 components of, 106–108
 counters, 434–439
 writing in Java, 109–114
 Pig
 parameterizing, 159–160
 reusing code, 160–161

Spark
 planning, 226
 scheduling, 226
 support for, 222
YARN
 killing, 288–289
 running, 96–99, 281–284
 scheduling, 293–298
architecture
 of Apache Ambari, 334–335
 of Cloudera Manager, 331–332
 of Flume, 64–67
 of HBase, 268–269
 of Kafka, 314–315
 of Oozie, 309
 of Spark, 225
 drivers, 226–227
 executors, 227
 of YARN, 279–281
archiving in HDFS, 402
ARRAY datatype, 187–188
ASF (Apache Software Foundation), 2
asymmetry of MapReduce, 89–90
atoms (Pig Latin), 128
attribute value pairs. *See* **key value pairs**
authentication, 54, 411
 in HUE, 255
 for web UIs, 407–409
authorization, 53, 411
 Hive, 169–171
 in HUE, 256
automating Hive
 custom scripts, 186–187

 parameterizing queries, 185–186
auxiliary services in YARN, 284
Avro format, 105, 318–319
AWS (Amazon Web Services)
 deploying Hadoop clusters, 38–41
 EC2, 38–39
 EMR, 40
 pricing, 40–41
 S3, 39

B

backing up NameNodes, 423
bags (Pig Latin), 128, 133
balancer utility, 361–362
Beeline, 168
Beeswax, 168
benchmarking Hadoop, 362–365
best practices
 data ingestion, 77–78
 debugging, 440
big data, history of, 1–2
Bigtable, 263–264
blocks in HDFS, 12, 46–48
 DataNodes, 49–50
 distribution of, 47
 failure recovery, 48
 replication of, 47–48
blueprints (Ambari), 339–340
box classes, 104
bucketing in Hive, 201–202
built-in functions
 in Hive, 175–176
 in Pig, 136

- starting and stopping services, 421–422
 - troubleshooting
 - Apache Ambari, 347–350
 - Cloudera Manager, 347–350
 - COGROUP statement (Pig), 144–145**
 - cogrouping multiple datasets in Pig, 144–145**
 - co-locating slave nodes, 21**
 - column encryption with Hive, 197**
 - columnar storage, 208–209**
 - ORC format, 210–211
 - Parquet format, 209–210
 - Combiner functions of MapReduce, 88–89, 117**
 - comments in Pig Latin, 131**
 - commercial Hadoop distributions, 5–6, 33–38, 42**
 - common properties**
 - for HDFS, 372–373
 - for YARN, 377
 - community function libraries in Pig, 157–158**
 - compiling MapReduce applications, 114–116**
 - complex datatypes in Hive, 187–190**
 - Complex Event Processing (CEP), 447–448**
 - concatenating multiple datasets in Pig, 153**
 - configuration, 367–368**
 - default values, 368–369
 - exclude files, 359–361
 - hadoop-env.sh file, 371
 - hadoop-metrics.properties file, 372
 - HBase, 383**
 - HDFS**
 - client properties, 376–377
 - common properties, 372–373
 - DataNode properties, 375–376
 - NameNode properties, 373–374
 - SecondaryNameNode properties, 374–375
 - Hive, 169, 178–179, 381**
 - HUE, 251–255**
 - include files, 357–359**
 - Job object and, 107**
 - log4j.properties file, 371**
 - Pig, 381–382**
 - precedence rules, 369–370**
 - Spark, 382**
 - YARN, 285–286**
 - common properties, 377
 - MapReduce properties, 379–380
 - NodeManager properties, 377–379
 - YARN CapacityScheduler, 298**
 - configuration management**
 - Apache Ambari, 345–346
 - Cloudera Manager, 342–344
 - consistency in HDFS, 54**
 - containers, 15, 96**
 - Context object, 112–113**
 - CONTEXT_NGRAMS function, 195**
 - control flow nodes, 306–307**
 - core-site.xml properties, 372–373**
 - counters, 434–439**
 - counting words, 92–95**
 - CQL (Cassandra Query Language), 273–274**
 - CREATE TABLE statement (Hive), 172**
 - cross joins in Pig, 154–155**
 - CROSS statement (Pig), 154–155**
 - custom connectors for Sqoop, 73**
 - custom scripts in Hive, 186–187**
 - Cutting, Doug, 1–2**
- ## D
- daemons, 11**
 - in HDFS, 21
 - log files for, 428–430
 - in YARN, 21, 279–281
 - DAGs (Directed Acyclic Graphs), 215**
 - data**
 - aggregating
 - in Hive, 176
 - in Pig, 143
 - analyzing in Hive, 175
 - distribution in HBase, 268–269
 - extracting with Regex SerDe, 191
 - filtering in Pig, 133, 150
 - grouping
 - in Hive, 176
 - in Pig, 141–142
 - ingesting
 - best practices, 77–78

- with Flume, 63–70
- with RESTful interfaces, 74–77
- with Sqoop, 70–74
- inspecting schema in Pig, 135
- joining in Hive, 176–177
- loading
 - in Hive, 174
 - in Pig, 131–133
- multiple datasets (Pig)
 - cogrouping, 144–145
 - concatenating, 153
 - cross joins, 154–155
 - joining, 145–151
 - removing duplicates, 154
 - splitting, 155–156
 - subtracting tuples, 154
- nested FOREACH statements
 - in Pig, 143–144
- ordering in Pig, 134
- outputting in Hive, 180–181
- projecting and transforming in Pig, 134
- running queries in Pig, 136–138
- data center, Hadoop integration in, 443–444**
- data ingestion utilities, 301–303**
 - Flume, 302–303. *See also* Flume
 - Sqoop, 302. *See also* Sqoop
- data locality, 2–3, 8, 229**
- data masking in Hive, 196–199**
- data processing. *See* HBase; Hive; MapReduce; Pig; Spark**
- data science. *See* advanced analytics use case**
- data storage and exchange formats, 318–319**
- data structures in Pig Latin, 128**
- data transport encryption, 410**
- data warehouse offload use case, 7, 445–446–447**
- databases**
 - conventional, 167
 - Hive, 169
- DataFrames, 234–236**
- DataFu, 158**
- DataNodes, 13, 49–50**
 - block distribution, 359
 - block failure recovery, 48
 - properties for HDFS, 375–376
- datatypes**
 - in Hadoop, 104
 - in Hive, 171, 187–190
 - in Pig Latin, 129–130
- debugging. *See* troubleshooting**
- decommissioning nodes from clusters, 359–361**
- deep storage, 444**
- default configuration values, 368–369**
- deleting**
 - directories in HDFS, 57
 - files in HDFS, 57
- deploying**
 - clusters
 - Apache Ambari, 337–340
 - in cloud, 38–41
 - Cloudera Manager, 33–34, 340
 - commercial Hadoop, 33–38, 42
 - Hortonworks, 35–36
 - MapR, 36–38
 - non-commercial Hadoop, 27–32
 - Hive, 169, 178–179
 - NameNode high availability, 393–398
- deployment modes, 18–20**
- deprecation of property names, 368**
- dereferencing operator, 143**
- DESCRIBE FUNCTION statement (Hive), 176**
- DESCRIBE statement**
 - in Hive, 172
 - in Pig, 135
- deserialization, 103**
- dfsadmin command, 422–423**
- dfs.blocksize property, 376**
- dfs.datanode.data.dir property, 375**
- dfs.datanode.du.reserved property, 375–376–376**
- dfs.namenode.checkpoint.dir property, 374–375**
- dfs.namenode.checkpoint.period property, 374–375**
- dfs.namenode.checkpoint.txns property, 374–375**
- dfs.namenode.edits.dir property, 373–374**
- dfs.namenode.name.dir property, 373–374**
- dfs.replication property, 376**
- Directed Acyclic Graphs (DAGs), 215**
- directories in HDFS**
 - deleting, 57

listing contents, 57

paths, 56

DISTINCT statement (Pig), 154

distributed computing, limitations of, 82

DistributedCache, 118–119

document stores, 265

downloading

files in HDFS, 57

Oozie, 311

Drill, 208, 216–217

drivers

MapReduce, 106–107, 109–111

Spark, 226–227

DROP PARTITION statement (Hive), 200

DStreams, 236–237, 312–313

DUMP statement (Pig), 138

duplicates, removing in Pig, 154

E

EC2 (Elastic Compute Cloud), 38–39

ecosystem projects, 4–5, 301.

See also names of specific ecosystem projects

data ingestion utilities, 301–303

data processing utilities, 303–304

data storage and exchange formats, 318–319

infrastructure and security, 315–318

machine learning, 319–323

NoSQL solutions, 305–306

notebook platforms, 323–325

search platforms, 306

stream processing, 311–315

workflow tools, 311

edits file, 54

Elastic Compute Cloud (EC2), 38–39

election analogy for MapReduce, 91–92

EMR (Elastic MapReduce), 40

encryption

data at rest, 411

data transport encryption, 410

enterprise integration of Hadoop, 443–444. *See also* use cases

environment variables for all users, 59

event lakes, 448

event processing use case, 7, 447–449

exclude files, configuring, 359–361

executing queries in Hive, 203–204

executors, Spark, 227

EXPLAIN statement (Hive), 203

EXPLODE function, 192

extending MapReduce, 120

extensions to Spark

GraphFrames, 239–240

GraphX, 239–240

MMLib, 238–239, 450

Spark Streaming, 236–237

SparkR, 237–238

SparkSQL, 234–236

external tables, internal versus, 173

extracting data with RegEx SerDe, 191

F

failover, types of, 392–393

failure recovery

HDFS blocks, 48

YARN, 283–284

FairScheduler, 293–295

fault tolerance, 12

MapReduce, 88

master and slave nodes, 42

with RDDs, 230–231

YARN, 283–284

feature engineering, 449–450

federation in HDFS, 398–401

fencing, 391–392

fields (Pig Latin), 128

FIFOScheduler, 293

file globs, 111

files in HDFS, 12, 46–48

deleting, 57

downloading, 57

ingesting, 56

reading and writing, 50–52, 60

FILTER statement (Pig), 133

filtering data in Pig, 133, 150

Flink, 208

Flume, 63–70, 302–303

architecture, 64–67

installing, 67–70

FOREACH statement (Pig), 134

- built-in functions, 136
- nesting, 143–144

fs.defaultFS property, 372**fsimage file, 54****full outer joins, 145****fully-distributed mode, 18****functions**

- built-in functions in Pig, 136
- text processing in Hive, 192–193–194
- UDFs. See UDFs (user-defined functions)

future of NoSQL, 275**G****GRANT statement (Hive), 170****graph processing with Spark, 239–240****graph stores, 265****GraphFrames, 239–240****GraphX, 239–240****GROUP ALL statement (Pig), 143****GROUP statement (Pig), 141–142****grouping**

- data
 - in Hive, 176
 - in Pig, 141–142
- multiple datasets in Pig, 144–145

grunt shell (Pig), 127–128**H****H₂O, 321–322, 450****Hadoop**

- Cassandra with, 274–275
- cluster architecture, 17–18
- commercial distributions, 5–6
- core components, 3–4, 9
- deployment modes, 18–20
- ecosystem projects, 4–5
- enterprise integration in data center, 443–444
- explained, 2–3
- HBase with, 272
- history of, 1–2
- installing, 26–27
 - in cloud, 38–41
 - Cloudera Manager, 33–34
 - commercial Hadoop, 33–38, 42
 - Hortonworks, 35–36
 - MapR, 36–38
 - non-commercial Hadoop, 27–32
- MapReduce in, 95–96
- requirements
 - hardware, 24–26
 - operating system, 23–24
 - software, 26
- use cases, 6–8
 - data warehouse offload, 445–446–447
 - event processing, 447–449
 - predictive analytics, 448–450

Hadoop Distributed File System (HDFS). See HDFS (Hadoop Distributed File System)**Hadoop User Environment (HUE). See HUE (Hadoop User Environment)****hadoop-env.sh file, 371****hadoop-metrics2.properties file, 433–434****hadoop-metrics.properties file, 372****hadoop.tmp.dir property, 377****hardware requirements, 24–26****hashing functions in Hive, 196–199****HAWQ, 208, 216****HBase, 265, 305**

- architecture, 268–269
- configuration, 383
- data model, 266
- data processing in, 266–268
- with Hadoop, 272
- HUE HBase browser interface, 249
- installing, 269–272

HDFS (Hadoop Distributed File System), 3–4, 9

- ACLs, 53–54
- administration, 422–424
- archiving, 402
- blocks, 12, 46–48
 - distribution of, 47
 - failure recovery, 48
 - replication of, 47–48
- caching, 401
- cluster processes, 11–14, 21

- DataNodes, 49–50
- NameNode, 48–49
- SecondaryNameNode, 55, 60
- configuration
 - client properties, 376–377
 - common properties, 372–373
 - DataNode properties, 375–376
 - NameNode properties, 373–374
 - SecondaryNameNode properties, 374–375
- consistency, 54
- daemons, 21
- directories
 - deleting, 57
 - listing contents, 57
 - paths, 56
- federation, 398–401
- files, 12, 46–48
 - deleting, 57
 - downloading, 57
 - ingesting, 56
 - reading and writing, 50–52, 60
- high availability, 55, 390–398
 - deploying, 393–398
 - failover types, 392–393
 - fencing, 391–392
- HttpFS, 76–77
- interacting via HUE, 244–245
- interfaces for, 56
- metadata, 12, 52–53, 60
 - on-disk structures, 54
 - permissions, 53–54, 406–407
 - principles of, 45–46
 - rack awareness, 387–389
 - recovery, 54
 - replication, 46–48
 - safe mode, 54–55
 - snapshots, 401–402
 - Trash directory settings, 57
 - WebHDFS, 74–76
- high availability**
 - in HDFS, 55, 390–398
 - deploying, 393–398
 - failover types, 392–393
 - fencing, 391–392
 - for YARN Resource Manager, 280
- history**
 - of big data, 1–2
 - of Hadoop, 1–2
- Hive, 165–166, 303–304**
 - administration, 425–426
 - authorization, 169–171
 - built-in functions, 175–176
 - CLI, 168–169
 - configuration, 381
 - conventional databases versus, 167
 - creating objects, 171–173
 - databases, 169
 - datatypes
 - complex, 187–190
 - simple, 171
 - deploying and configuring, 169, 178–179
 - HUE Hive editor, 246–247
 - InputFormats, 173–174
 - metastore, 166–168
 - OutputFormats, 173–174
 - query execution, 203–204
 - query optimization
 - bucketing, 201–202
 - partitioning, 199–201
 - SerDes, 173–174
 - tables, 169
 - analyzing data, 175
 - grouping and aggregating data, 176
 - internal versus external, 173
 - joining data, 176–177
 - loading data, 174
 - outputting data, 180–181
 - text processing
 - data masking and hashing, 196–199
 - functions for, 192–193
 - regular expressions, 190–192
 - sentiment analysis, 194–196
- HiveQL, 165, 175**
- HiveServer 2, 168**
- Hortonworks, 6, 35–36**
- HttpFS, 76–77**
- HUE (Hadoop User Environment), 243–244, 258–260**
 - advantages of, 244
 - authentication, 255
 - authorization, 256

configuring, 251–255
 HBase browser interface, 249
 HDFS file browser interface,
 244–245
 Hive editor interface, 246–247
 Impala query editor, 248
 installing, 251
 logging in, 256–258
 Oozie editor, 250, 310
 Pig editor, 247–248
 Solr interface, 249–250
 Sqoop interface, 251
 YARN management, 245
hue.ini file, 251–255

I

identifiers in Pig Latin, 128–129
ILLUSTRATE statement (Pig), 135
immutability, 46
Impala, 208, 211–214, 248
IMPORT statement (Pig), 161
**include files, configuring,
 357–359**
infrastructure projects, 315–318
ingesting
 data
 best practices, 77–78
 with Flume, 63–70
 with RESTful interfaces,
 74–77
 with Sqoop, 70–74
 files in HDFS, 56
inner joins, 145
InputFormats, 105, 173–174

**input/output types in Spark,
 224–225**
InputPaths, 111
INSERT statement (Hive), 181
inspecting schemas in Pig, 135
installing
 Apache Ambari, 335, 351
 Cloudera Manager,
 332–333, 351
 Flume, 67–70
 Hadoop, 26–27
 in cloud, 38–41
 Cloudera Manager, 33–34
 commercial Hadoop,
 33–38, 42
 Hortonworks, 35–36
 MapR, 36–38
 non-commercial Hadoop,
 27–32
 HBase, 269–272
 HUE (Hadoop User
 Environment), 251
 Pig, 126
 Spark, 232–234
**internal tables, external
 versus, 173**
IoT (Internet of Things), 448
IPython, 324

J

**Java classes, naming
 conventions, 106**
**Java Database Connectivity
 (JDBC), 71, 73**

Java MapReduce API
 applications
 compiling and packaging,
 114–116
 writing, 109–114
 Combiner functions, 117
 DistributedCache, 118–119
 Hadoop datatypes, 104
 InputFormats, 105
 OutputFormats, 106
 Partitioners, 117–118
 SequenceFiles, 105
 serialization, 103
Java Virtual Machine (JVM), 19
**JBOD (just a bunch of disks),
 25, 42**
**JDBC (Java Database
 Connectivity), 71, 73**
Job object, configuration and, 107
JOIN statement
 in Hive, 176
 in Pig, 149–150
joining
 data in Hive, 176–177
 multiple datasets in Pig,
 145–151
joins, types of, 145
Jupyter, 324–325
JVM (Java Virtual Machine), 19

K

Kafka, 313–315, 447
Kerberos, 411–414
key value pairs, 83–84, 114

- datatypes, 104
 - serialization, 103
 - key value stores, 265**
 - keywords in Pig Latin, 129**
 - killing**
 - applications in YARN, 288–289
 - Hive queries, 203
 - Knox, 415–416**
- L**
- LATERAL VIEW clause, 193**
 - lazy evaluation in Spark, 231–232**
 - left outer joins, 145**
 - LEFT SEMI JOIN statement (Hive), 177**
 - LEVENSHTEIN function, 196**
 - libraries in Pig, 157–158**
 - linear scalability, 355–356**
 - Lisp, 84–85**
 - listing directory contents in HDFS, 57**
 - listings**
 - accessing files in DistributedCache, 119
 - accessing HBase from PySpark, 272
 - accessing logs for completed applications, 430–431
 - accessing MRJobHistory Server via REST, 292
 - accessing SparkSQL DataFrames, 235–236
 - actions in Spark, 231
 - ADD FILE and TRANSFORM statements, 187
 - ADD PARTITION statement, 200
 - adding files to DistributedCache from command line, 119
 - adding Hadoop environment variables for all users, 59
 - adjusting daemon log level, 429
 - adjusting log level programmatically, 430
 - Ambari blueprints, 339–340
 - Apache DataFu function library, 158
 - ARRAY datatype, 188
 - array() function, 188
 - assigning cluster ID when formatting NameNode, 399
 - assigning HDFS object permissions and ownership, 53
 - AvroMapper, 318–319
 - backing up NameNode's fsimage file, 424
 - balancer utility, 362
 - built-in functions in Pig, 136
 - changing Hive database contexts, 169
 - changing object permissions, 407
 - changing ownership of object in HDFS, 407
 - COGROUP statement, 145
 - comments in Pig, 131
 - common built-in functions included with Pig, 136
 - configuring CapacityScheduler, 296–297
 - configuring client side mount points for HDFS federation, 400
 - configuring connection to shared Hive metastore, 381
 - configuring FairScheduler, 295
 - configuring HA fencing, 392
 - configuring Hive authorization, 170
 - configuring SSL transport security for YARN, 287
 - CONTEXT_NGRAMS function, 195
 - CREATE TABLE AS SELECT, 174
 - CREATE TABLE statement, 172
 - CREATE TABLE with bucketing, 201
 - CREATE TABLE with partitioning, 199
 - creating and updating Java MapReduce counters, 435
 - creating directory snapshot, 402
 - creating Hadoop archive, 402
 - creating keyspace and table in Cassandra using cqlsh, 274
 - creating roles in Hive, 170
 - creating snapshottable directory in HDFS, 402
 - creating table and inserting data in HBase, 267
 - CROSS statement, 155
 - custom Partitioner, 118
 - custom transform script written in Python, 186–187

- declaring Combiner in Driver class, 117
- declaring Partitioner in Driver class, 118
- defining and calling Pig macro, 160–161
- DESCRIBE FUNCTION statement, 176
- DESCRIBE statement, 135, 172
- dfs.blocksize configuration property, 376
- dfs.datanode.data.dir configuration property, 375
- dfs.datanode.du.reserved configuration property, 376
- dfs.namenode.name.dir configuration property, 374
- dfs.replication configuration property, 376
- DISTINCT statement, 154
- DROP PARTITION statement, 200
- enabling Capacity Scheduler, 296
- enabling FairScheduler, 294
- enabling log aggregation, 379
- enabling rack topology script, 389
- enabling the capability to kill running jobs using UI, 288
- encryption and decryption in Hive, 197
- example log events, 428
- EXPLAIN statement, 203
- EXPLODE function, 192
- file system ACLs, 406
- FILTER statement, 133
- final output (word count algorithm), 95
- <final> tag, 370
- Flume agent channel configuration, 67
- Flume agent sink configuration, 66
- Flume agent source configuration, 66
- FOREACH statement, 134
- fs.defaultFS setting, 372
- GRANT statement, 170
- GROUP ALL statement, 143
- GROUP statement, 142
- grouping and aggregating data in Hive, 176
- Hadoop configuration needed to join cluster, 357
- hadoop-metrics2.properties file, 433–434
- hadoop.tmp.dir configuration property, 377
- hbase-site.xml configuration file, 383
- Hive built-in functions, 175
- hue.ini file, 252–254
- ILLUSTRATE statement, 135
- IMPORT statement, 161
- importing all tables from database using Sqoop, 73
- input to map task, 93
- INSERT statement, 181
- installing ambari-server package using EPEL, 335
- intermediate data sent to Reducer, 94
- item based recommender in Mahout, 320
- JOIN statement, 149–150, 176
- killing Hive query, 203
- killing YARN application using yarn command, 290
- LATERAL VIEW clause, 193
- leaving safemode, 423
- LEFT SEMI JOIN statement, 177
- LEVENSHTEIN function, 196
- LOAD statement, 174
- local execution in Hive, 204
- log4j.properties file, 371, 427
- manual failover, 392
- MAP datatype, 190
- mapreduce.framework.name configuration property, 380
- mapreduce.jobhistory.address configuration property, 380
- MASK function, 197
- MD5 function, 197
- MSCK REPAIR TABLE command, 201
- multiple Pig parameters in text file, 160
- nested FOREACH statements, 144
- NGRAMS function, 195
- ORDER statement, 134
- OVERWRITE option, 174
- PageRank algorithm in Spark, 240
- parameterized HiveQL query, 185
- Parquet storage, 210
- PARSE_URL function, 198

- PARSE_URL_TUPLE
 - function, 199
- passing parameters to Pig script, 160
- passing Pig parameters using text file, 160
- passing shell command return values to Pig script, 160
- performing snapshot diff, 402
- Pig dryrun option, 161
- Pig identifiers, 129
- Pig program, 126
- Pig program variables sourced from parameters, 159
- Pig properties file, 382
- Pig reserved keywords, 129
- Pig UDF written in Jython, 157
- PiggyBank function library, 158
- PigStorage LOAD function, 132
- POSEXPLODE function, 193
- rack topology script, 389
- RDDs in Spark, 228
- reading MapReduce counters, 435
- referencing Hive objects, 169
- refreshing CapacityScheduler configuration, 297
- Regex SerDe, 191
- REGEXP_EXTRACT
 - function, 191
- REGEXP_REPLACE
 - function, 192
- registering and using Pig UDF, 157
- REST access to HDFS using WebHDFS, 74
- RESTful HDFS access using HttpFS, 77
- REVOKE statement, 170
- running Hive queries in batch mode, 169
- sample datasets, 142
- sample Oozie workflow document, 307–308
- scanning HBase table, 267
- schematool usage to initialize metastore schema, 425
- securing NameNode's web UI using HTTPS, 408
- SELECT statement, 175
- selecting ARRAY elements, 188
- selecting MAP values, 190
- selecting STRUCT elements, 189
- SENTENCES function, 194
- SET statement, 186
- setting Hive variable via the command line, 186
- setting Spark configuration properties programmatically, 382
- SHOW PARTITIONS
 - statement, 200
- SOUNDEX function, 196
- Spark configuration properties, 382
- Spark MLlib, 320–321
- specifying log4j.properties file for application, 428
- specifying number of Reduce tasks from command line, 110
- SPLIT function, 192
- SPLIT statement, 155–156
- Sqoop tools, 72–73
- Sqoop2 shell, 302
- starting services in Hadoop, 422
- STREAM statement, 159
- STRUCT datatype, 189
- submitting Oozie workflow using command line client, 309
- submitting Spark application, 224
- submitting streaming MapReduce job, 122
- SUBTRACT statement, 154
- SUM function, 143
- supplying schemas for bags in Pig, 132–133
- TABLESAMPLE operator, 202
- Timeline Server
 - REST API, 293
- ToolRunner command line generic options, 110
- training a decision tree model with Spark MLlib, 239
- transformations in Spark, 231
- transport security for intermediate data, 410
- UNION statement, 153
- updating cell in HBase table, 268
- updating data in Cassandra table using Spark, 275

- valid and invalid Pig identifiers, 129
- ViewFS, 400
- viewing YARN logs using yarn command, 290–291
- web log schema using RegEx SerDe, 198
- word count mapper in Python, 121
- word count reducer in Python, 121
- WordCount Driver class, 109
- WordCount Mapper class, 111
- WordCount Reducer class, 113
- yarn command, 289
- yarn command usage to control YARN applications, 425
- YARN container related configuration parameters, 282
- yarn.app.mapreduce.
 - am.staging-dir configuration property, 380
- yarn.nodemanager.aux-services configuration property, 378
- yarn.nodemanager.local-dirs configuration property, 378
- yarn.nodemanager.log-dirs configuration property, 379
- LOAD statement (Hive), 174**
- loading data
 - in Hive, 174
 - in Pig, 131–133
- local mode, running Hive, 203–204
- LocalJobRunner mode, 19–20

- log aggregation in YARN, 290–291, 379
- log files
 - within applications, 430–431
 - for daemons, 428–430
 - Log4j, 426–427
 - log4j.properties file, 371, 427–428
- Log4j, 426–427
- log4j.properties file, 371, 427–428
- logging in to HUE, 256–258
- Lucene, 306
- LVM (Logical Volume Manager), 24

M

- machine learning, 319–323
 - with H₂O, 321–322
 - with Mahout, 320
 - predictive analytics use case, 448–450
 - Presto, 323
 - with Spark, 238–239, 320–321
- macros in Pig, 160–161
- Mahout, 320, 450
- MAP datatype, 189–190
- Map phase of MapReduce, 85–86, 98
- Map-only MapReduce applications, 90–91
- Mappers (in MapReduce), 107–108, 111–113
- MapR, 6, 36–38

- mapred-site.xml file, 379
- MapReduce. *See also* Java MapReduce API
 - applications
 - compiling and packaging, 114–116
 - components of, 106–108
 - writing in Java, 109–114
 - asymmetry, 89–90
 - Combiner functions, 88–89, 117
 - counters, 434–439
 - design goals of, 83
 - election analogy, 91–92
 - EMR, 40
 - extending, 120
 - fault tolerance, 88
 - Google whitepapers on, 81
 - in Hadoop, 95–96
 - history with YARN, 291–293
 - key value pairs, 83–84
 - LocalJobRunner mode, 19–20
 - Map phase, 85–86, 98
 - Map-only applications, 90–91
 - motivation for, 82
 - Partitioners, 86–87, 117–118
 - programming model, 84–85
 - properties for YARN, 379–380
 - Reduce phase, 87–88, 98–99
 - Shuffle and Sort phase, 87, 98
 - speculative execution, 89–90
 - Sqoop and, 71–72
 - word count algorithm, 92–95
- MapReduce Streaming API, 120–122

mapreduce.framework.name property, 380

mapreduce.jobhistory.address property, 380

MASK function, 197

master nodes

- fault tolerance, 42
- hardware requirements, 24

master-slave cluster architecture, 12

mathematical operators in Pig Latin, 130–131

MBeans, 432–434

metadata in HDFS, 12, 52–53, 60

metastore (Hive), 166–168, 425–426

metrics framework, 432–434

MLib, 238–239, 320–321, 450

monitoring clusters

- Apache Ambari, 347–350
- Cloudera Manager, 347–350
- counters, 434–439
- MBeans, 432–434

MRJobHistory Server, 291–292

MSCK REPAIR TABLE command (Hive), 201

multiple datasets

- cogrouping in Pig, 144–145
- concatenating in Pig, 153
- cross joins in Pig, 154–155
- joining in Pig, 145–151
- removing duplicates in Pig, 154
- splitting in Pig, 155–156
- subtracting tuples in Pig, 154

N

name value pairs. *See key value pairs*

NameNodes, 12–13, 48–49

- backing up, 423
- federation, 398–401
- high availability, 390–398
 - deploying, 393–398
 - failover types, 392–393
 - fencing, 391–392
- metadata, 52–53, 60
- properties for HDFS, 373–374
- safe mode, 54–55
- as single point of failure, 55

nameservices, 394

naming conventions

- bags (Pig Latin), 133
- Java classes, 106
- in Pig Latin, 128–129, 131

nested FOREACH statements (Pig), 143–144

networks, hardware requirements, 26

n-grams, 156–157, 194–195

NGRAMS function, 194–195

NodeManager, 16, 96, 281

- failure recovery, 283
- properties for YARN, 377–379

nodes, 4

- action nodes, 307
- adding to cluster, 356–359
- control flow nodes, 306–307
- decommissioning from cluster, 359–361

- master nodes
 - fault tolerance, 42
 - hardware requirements, 24
- slave nodes
 - co-locating, 21
 - fault tolerance, 42
 - hardware requirements, 25

non-commercial Hadoop installing, 27–32

normalizing text, 195

NoSQL, 305–306. *See also* Cassandra; HBase

- Bigtable and, 263–264
- characteristics of, 264–265
- future of, 275
- implementations of, 275
- types of systems, 265

notebook platforms, 323–325

O

object identifiers in Pig Latin, 128–129

objects (Hive), creating, 171–173

ODPi (Open Data Platform initiative), 6

on-disk structures in HDFS, 54

on-platform analytics, 450

Oozie

- architecture, 309
- downloading, 311
- HUE Oozie editor, 250, 310
- workflows, 306–308, 309–310

Open Data Platform initiative (ODPi), 6

operating system requirements, 23–24

operators

- mathematical operators in Pig Latin, 130–131
- relational operators in Pig Latin, 130

optimizing

- joins in Pig, 147
- queries in Hive, 199–202

ORC (Optimized Row Columnar) format for columnar storage, 210–211

ORDER statement (Pig), 134

ordering data in Pig, 134

outer joins, 145

OutputFormats, 106, 173–174

outputting data in Hive, 180–181

P

packaging MapReduce applications, 114–116

PageRank, 240

parallel execution in Hive, 203

parameterizing

- applications in Pig, 159–160
- queries in Hive, 185–186

Parquet format for columnar storage, 209–210

parsing URLs in Hive, 198–199

Partitioners, 86–87, 117–118

partitioning in Hive, 199–201

paths for directories, 56

performance benchmarking Hadoop, 362–365

perimeter security, 414–416

permissions in HDFS, 53–54, 406–407

persistence with RDDs, 229

persistent staging offload, 445–446

Pig, 125–126, 304

- applications
 - parameterizing, 159–160
 - reusing code, 160–161
- built-in functions, 136
- configuration, 381–382
- data
 - aggregating, 143
 - filtering, 133, 150
 - grouping, 141–142
 - inspecting schema, 135
 - loading, 131–133
 - nested FOREACH statements, 143–144
 - ordering, 134
 - projecting and transforming, 134
 - running queries, 136–138
- grunt shell, 127–128
- HUE Pig editor, 247–248
- installing, 126
- modes, 127
- multiple datasets
 - cogrouping, 144–145
 - concatenating, 153
 - cross joins, 154–155
 - joining, 145–151
 - removing duplicates, 154
 - splitting, 155–156
 - subtracting tuples, 154

projecting data in Pig

- STREAM statement, 158–159
- UDFs
 - community function libraries, 157–158
 - registering, 157
 - writing, 156–157
- WordCount example, 151

Pig Latin, 125

- comments, 131
- data structures, 128
- mathematical operators, 130–131
- object identifiers, 128–129
- program flow, 128
- relational operators, 130
- simple datatypes, 129–130
- statements, 131

PiggyBank, 157–158

Pivotal HAWQ, 208, 216

planning applications in Spark, 226

polices, access control, 416–418

POSEXPLODE function, 193

precedence rules in configuration, 369–370

predictive analytics use case, 7–8, 448–450

preemption, 295

Presto, 208, 323

pricing AWS, 40–41

processing data. *See* HBase; Hive; MapReduce; Pig; Spark

programming interfaces to Spark, 222–225

programming model of MapReduce, 84–85

projecting data in Pig, 134

properties

- client properties for HDFS, 376–377
- common properties
 - for HDFS, 372–373
 - for YARN, 377
- DataNode properties for HDFS, 375–376
- MapReduce properties for YARN, 379–380
- NameNode properties for HDFS, 373–374
- NodeManager properties for YARN, 377–379
- SecondaryNameNode properties for HDFS, 374–375
- property names, deprecation, 368
- pseudo-distributed mode, 19
- PySpark, 223

Q**queries**

- custom scripts in Hive, 186–187
- executing in Hive, 203–204
- optimizing in Hive
 - bucketing, 201–202
 - partitioning, 199–201
- parameterizing in Hive, 185–186
- running in Pig, 136–138

R

- rack awareness in HDFS, 387–389
- Ranger, 417
- RBAC (role-based access control), 416–418
- RDDs (Resilient Distributed Datasets), 227–229
 - actions, 231
 - data locality, 229
 - fault tolerance, 230–231
 - lineage, 229–230
 - persistence and re-use, 229
 - transformations, 231
- reading files in HDFS, 50–52
- rebalancing clusters, 361–362
- RecordReader objects, InputFormats, 105
- records in MapReduce, 83–84
- recovery in HDFS, 54
- Reduce phase of MapReduce, 87–88, 98–99
- Reducers (in MapReduce), 108, 113–114
- Regex SerDe, 191
- REGEXP_EXTRACT function, 191
- REGEXP_REPLACE function, 192
- registering UDFs in Pig, 157
- regular expressions in Hive, 190–192
- relational operators in Pig Latin, 130
- relations (Pig Latin), 128
- removing
 - duplicates in Pig, 154
 - tuples in Pig, 154

replication in HDFS, 46–48**requirements**

- hardware, 24–26
- operating system, 23–24
- software, 26

Resilient Distributed Datasets (RDDs). See RDDs (Resilient Distributed Datasets)**ResourceManager, 15, 96, 280, 284****ResourceManager UI, 286–289****ResourceRequests, 281–283****resources for information, 440****RESTful interfaces, data ingestion with, 74–77****re-use with RDDs, 229****reusing code in Pig, 160–161****REVOKE statement (Hive), 170****right outer joins, 145****role-based access control (RBAC), 416–418****running**

- applications on YARN, 96–99, 281–284
- Hive in local mode, 203–204
- queries in Pig, 136–138

S**S3 (Simple Storage Service), 39****safe mode in HDFS, 54–55****sbin directory scripts, 422****Scala, 224****scaling Hadoop**

- adding nodes to cluster, 356–359

- benchmarking, 362–365
- decommissioning nodes from
 - cluster, 359–361
- linear scalability, 355–356
- rebalancing clusters, 361–362
- when to scale, 356
- scheduling applications**
 - in Spark, 226
 - in YARN, 293–298
- schema-on-read systems, 3**
- schema-on-write systems, 3**
- schemas, inspecting in Pig, 135**
- scripts, custom scripts in Hive, 186–187**
- search platforms, 306**
- SecondaryNameNode, 13, 55, 60, 374–375**
- security**
 - Apache Ambari, 351
 - authentication, 411
 - authorization, 411
 - Cloudera Manager, 351
 - ecosystem projects, 315–318
 - encryption
 - data at rest, 411
 - data transport encryption, 410
 - HDFS permissions, 406–407
 - Kerberos, 411–414
 - perimeter security, 414–416
 - RBAC (role-based access control), 416–418
 - for web UIs, 407–409
- SELECT statement (Hive), 175**
- SENTENCES function, 194**
- sentiment analysis in Hive, 194–196**
- Sentry, 418**
- SequenceFiles, 105**
- SerDes (Serializer/Deserializer), 173–174, 191**
- serialization, 103**
- service management**
 - Apache Ambari, 344–345
 - Cloudera Manager, 340–342
- services**
 - starting, 421–422
 - stopping, 421–422
- set operations (Pig)**
 - CROSS statement, 154–155
 - DISTINCT statement, 154
 - SPLIT statement, 155–156
 - SUBTRACT statement, 154
 - UNION statement, 153
- SET statement (Hive), 186**
- severity levels (Log4j), 426**
- shared nothing, 3, 9**
- SHOW PARTITIONS statement (Hive), 200**
- shuffle, 232**
- Shuffle and Sort phase of MapReduce, 87, 98**
- simple datatypes. See datatypes**
- Simple Storage Service (S3), 39**
- single point of failure (SPOF), 55**
- sinks (Flume agents), 66–67**
- slave nodes**
 - co-locating, 21
 - fault tolerance, 42
 - hardware requirements, 25
- snapshots in HDFS, 401–402**
- software distribution with Cloudera Manager, 332**
- software requirements, 26**
- Solr, 249–250, 306**
- SOUNDEX function, 196**
- sources (Flume agents), 65–66**
- Spark, 279–280, 304**
 - accumulators, 436
 - application support, 222
 - architecture, 225
 - drivers, 226–227
 - executors, 227
 - configuration, 382
 - extensions
 - GraphFrames, 239–240
 - GraphX, 239–240
 - MLlib, 238–239, 320–321, 450
 - Spark Streaming, 236–237
 - SparkR, 237–238
 - SparkSQL, 234–236
 - installing, 232–234
 - lazy evaluation, 231–232
 - programming interfaces, 222–225
 - RDDs, 227–229
 - actions, 231
 - data locality, 229
 - fault tolerance, 230–231
 - lineage, 229–230
 - persistence and re-use, 229
 - transformations, 231
- Spark Streaming, 236–237, 312–313**
- SparkContext, 226**
- SparkR, 237–238**
- SparkSQL, 234–236**

- spark-submit command, 224
- speculative execution of MapReduce, 89–90
- SPLIT function, 192
- SPLIT statement (Pig), 155–156
- splitting multiple datasets in Pig, 155–156
- SPOF (single point of failure), 55
- SQL on Hadoop, 304
 - Apache Drill, 216–217
 - Apache HAWQ, 216
 - explained, 207–208
 - Impala, 211–214
 - Tez, 214–216
- Sqoop, 70–74, 302
 - custom connectors, 73
 - HUE Sqoop interface, 251
 - MapReduce and, 71–72
 - Sqoop2 (Sqoop-as-a-Service), 73–74, 302
 - tools, 72–73
- Sqoop2 (Sqoop-as-a-Service), 73–74, 302
- SSL security for YARN, 287
- StandbyNameNode, 13
- starting services, 421–422
- statements in Pig Latin, 131
- stopping services, 421–422
- STORE command (Pig), 138
- Storm, 311–312
- stream processing, 311–315
 - Kafka, 313–315
 - Spark Streaming, 312–313
 - Storm, 311–312
- STREAM statement (Pig), 158–159
- STRUCT datatype, 189

- SUBTRACT statement (Pig), 154
- SUM function (Pig), 143
- symbolic notation for permissions, 406

T

- tables (Hive), 169
 - analyzing data, 175
 - grouping and aggregating data, 176
 - internal versus external, 173
 - joining data, 176–177
 - loading data, 174
 - outputting data, 180–181
- TABLESAMPLE operator (Hive), 202
- task failure, YARN, 283
- Terasort, 363–365
- text processing in Hive
 - data masking and hashing, 196–199
 - functions for, 192–193–194
 - regular expressions, 190–192
 - sentiment analysis, 194–196
- TextInputFormat, 93, 94
- Tez, 208, 214–216
- Thrift, 319
- tiered storage, 444
- Timeline Server, 292–293
- tokenization, 93
- tools in Sqoop, 72–73
- TPC (Transaction Processing Performance Council), 363
- TRANSFORM operator (Hive), 186–187

- transforming
 - data in Pig, 134
 - RDDs, 231
- Transparent Encryption, 411
- Trash directory settings, 57
- troubleshooting
 - best practices for, 440
 - clusters
 - Apache Ambari, 347–350
 - Cloudera Manager, 347–350
 - log files for
 - within applications, 430–431
 - daemon logs, 428–430
 - Log4j, 426–427
 - log4j.properties file, 427–428
 - resources for information, 440
 - with web UIs, 431–432
- tuples (Pig Latin), 128, 154

U

- UDFs (user-defined functions)
 - community function libraries in Pig, 157–158
 - registering in Pig, 157
 - writing in Pig, 156–157
- UNION statement (Pig), 153
- uploading. *See* ingesting
- URL parsing in Hive, 198–199
- use cases, 6–8
 - data warehouse offload, 445–446–447

event processing, 447–449
 predictive analytics, 448–450
user-defined functions. See UDFs (user-defined functions)

V

ViewFS, 400

W

web log parsing in Hive, 198–199

web UIs

security for, 407–409
 troubleshooting with, 431–432

WebHDFS, 74–76

word count algorithm, 92–95

workflows

in Oozie, 306–308, 309–310
 tools for, 311

Writable interface, 104

WritableComparable interface, 104

writing

files in HDFS, 50–52, 60
 MapReduce applications, 109–114
 Driver code, 109–111
 Mapper code, 111–113
 Reducer code, 113–114
 UDFs in Pig, 156–157

Y

YARN (Yet Another Resource Negotiator), 3–4, 9

administration, 424–425
 application scheduling, 293–298
 CLI usage, 289–290
 cluster processes, 14–17, 21, 279–281
 configuration, 285–286
 common properties, 377
 MapReduce properties, 379–380
 NodeManager properties, 377–379
 daemons, 21
 job management via HUE, 245

log aggregation, 290–291, 379
 MapReduce history, 291–293
 ResourceManager UI, 286–289
 running applications on, 96–99, 281–284

yarn application commands, 424

yarn rmadm commands, 424

yarn.app.mapreduce.am.staging-dir property, 380

yarn.nodemanager.aux-services property, 378

yarn.nodemanager.local-dirs property, 377–378

yarn.nodemanager.log-dirs property, 378–379

yarn.resourcemanager.hostname property, 377

yarn-site.xml properties, 377

Z

Zeppelin, 323–324

ZooKeeper, 315–318