2

Deploying Spark

The value of an idea lies in the using of it.

Thomas A. Edison, American inventor

In This Chapter:

- Overview of the different Spark deployment modes
- How to install Spark
- The contents of a Spark installation
- Overview of the various methods available for deploying Spark in the cloud

This chapter covers the basics of how Spark is deployed, how to install Spark, and how to get Spark clusters up and running. It discusses the various deployment modes and schedulers available for Spark clusters, as well as options for deploying Spark in the cloud. If you complete the installation exercises in this chapter, you will have a fully functional Spark programming and runtime environment that you can use for the remainder of the book.

Spark Deployment Modes

There are several common deployment modes for Spark, including the following:

- Local mode
- Spark Standalone
- Spark on YARN (Hadoop)
- Spark on Mesos

Each deployment mode implements the Spark runtime architecture—detailed in Chapter 3, "Understanding the Spark Cluster Architecture"—similarly, with differences only in the way resources are managed across one or many nodes in the computing cluster.

In the case of deploying Spark using an external scheduler such as YARN or Mesos, you need to have these clusters deployed, whereas running Spark in Local mode or using the Spark Standalone scheduler removes dependencies outside Spark.

All Spark deployment modes can be used for interactive (shell) and non-interactive (batch) applications, including streaming applications.

Local Mode

Local mode allows all Spark processes to run on a single machine, optionally using any number of cores on the local system. Using Local mode is often a quick way to test a new Spark installation, and it allows you to quickly test Spark routines against small datasets.

Listing 2.1 shows an example of submitting a Spark job in local mode.

Listing 2.1 **Submitting a Spark Job in Local Mode**

Click here to view code image

```
$SPARK_HOME/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master local \
$SPARK_HOME/examples/jars/spark-examples*.jar 10
```

You specify the number of cores to use in Local mode by supplying the number in brackets after the `local` directive. For instance, to use two cores, you specify `local[2]`; to use all the cores on the system, you specify `local[*]`.

When running Spark in Local mode, you can access any data on the local filesystem as well as data from HDFS, S3, or other filesystems, assuming that you have the appropriate configuration and libraries available on the local system.

Although Local mode allows you to get up and running quickly, it is limited in its scalability and effectiveness for production use cases.

Spark Standalone

Spark Standalone refers to the built-in, or "standalone," scheduler. We will look at the function of a scheduler, or cluster manager, in more detail in Chapter 3.

The term *standalone* can be confusing because it has nothing to do with the cluster topology, as might be interpreted. For instance, you can have a Spark deployment in Standalone mode on a fully distributed, multi-node cluster; in this case, *Standalone* simply means that it does not need an external scheduler.

Multiple host processes, or services, run in a Spark Standalone cluster, and each service plays a role in the planning, orchestration, and management of a given Spark application running on the cluster. Figure 2.1 shows a fully distributed Spark Standalone reference cluster topology. (Chapter 3 details the functions that these services provide.)

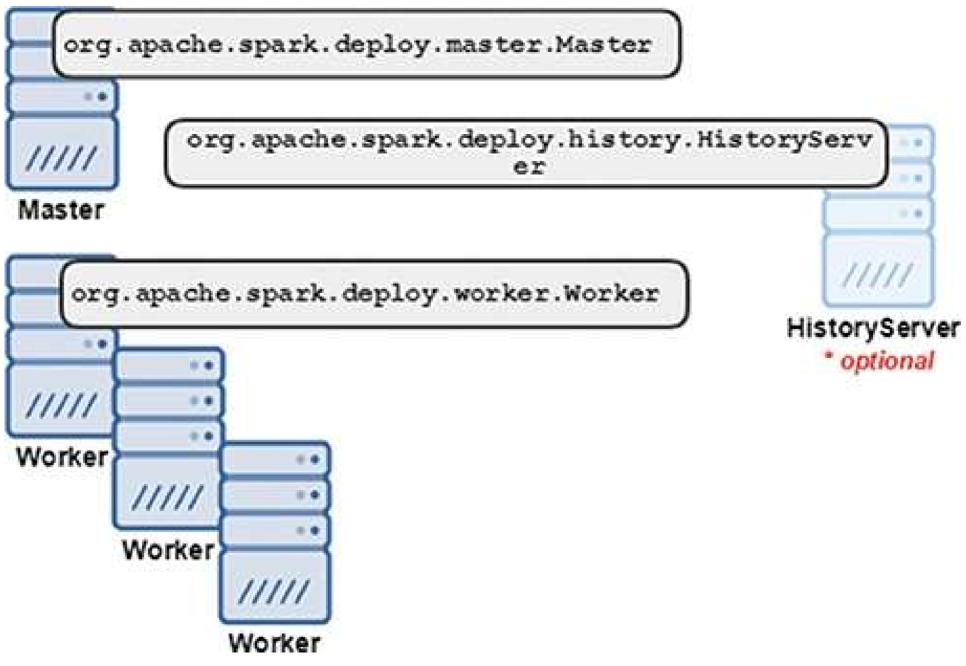

org.apache.spark.deploy.master.Master
Master
org.apache.spark.deploy.history.HistoryServer
HistoryServer
* optional
org.apache.spark.deploy.worker.Worker
Worker
Worker
Worker

Figure 2.1 Spark Standalone cluster.

You can submit applications to a Spark Standalone cluster by specifying `spark` as the URI scheme, along with the designated host and port that the Spark Master process is running on. Listing 2.2 shows an example of this.

Listing 2.2 **Submitting a Spark Job to a Spark Standalone Cluster**

Click here to view code image

```
$SPARK_HOME/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://mysparkmaster:7077 \
$SPARK_HOME/examples/jars/spark-examples*.jar 10
```

With Spark Standalone, you can get up and running quickly with few dependencies or environmental considerations. Each Spark release includes everything you need to get started, including the binaries and configuration files for any host to assume any specified role in a Spark Standalone cluster. Later in this chapter you will deploy your first cluster in Spark Standalone mode.

Spark on YARN

As introduced in Chapter 1, “Introducing Big Data, Hadoop, and Spark,” the most common deployment method for Spark is using the YARN resource management framework provided with Hadoop. Recall that YARN is the Hadoop core component that allows you to schedule and manage workloads on a Hadoop cluster.

According to a Databricks annual survey (see https://databricks.com/resources/type/infographic-surveys), YARN and standalone are neck and neck, with Mesos trailing behind.

As first-class citizens in the Hadoop ecosystem, Spark applications can be easily submitted and managed with minimal incremental effort. Spark processes such as the Driver, Master, and Executors (covered in Chapter 3) are hosted or facilitated by YARN processes such as the ResourceManager, NodeManager, and ApplicationMaster.

The `spark-submit`, `pyspark`, and `spark-shell` programs include command line arguments used to submit Spark applications to YARN clusters. Listing 2.3 provides an example of this.

Listing 2.3 **Submitting a Spark Job to a YARN Cluster**

Click here to view code image

```
$SPARK_HOME/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode cluster \
$SPARK_HOME/examples/jars/spark-examples*.jar 10
```

There are two cluster deployment modes when using YARN as a scheduler: `cluster` and `client`. We will distinguish between the two in Chapter 3 when we look at the runtime architecture for Spark.

Spark on Mesos

Apache Mesos is an open source cluster manager developed at University of California, Berkeley; it shares some of its lineage with the creation of Spark. Mesos is capable of scheduling different types of applications, offering fine-grained resource sharing that results in more efficient cluster utilization. Listing 2.4 shows an example of a Spark application submitted to a Mesos cluster.

Listing 2.4 **Submitting a Spark Job to a Mesos Cluster**

Click here to view code image

```
$SPARK_HOME/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master mesos://mesosdispatcher:7077 \
--deploy-mode cluster \
--supervise \
--executor-memory 20G \
--total-executor-cores 100 \
$SPARK_HOME/examples/jars/spark-examples*.jar 1000
```

This book focuses on the more common schedulers for Spark: Spark Standalone and YARN. However, if you are interested in Mesos, a good place to start is http://mesos.apache.org.

Preparing to Install Spark

Spark is a cross-platform application that can be deployed on the following operating systems:

- Linux (all distributions)
- Windows
- Mac OS X

Although there are no specific hardware requirements, general Spark instance hardware recommendations are as follows:

- 8 GB or more of memory (Spark is predominantly an in-memory processing framework, so the more memory the better.)
- Eight or more CPU cores
- 10 GB or greater network speed
- Sufficient local disk space for storage, if required (SSD is preferred for RDD disk storage. If the instance is hosting a distributed filesystem such as HDFS, then a JBOD configuration of multiple disks is preferred. JBOD stands for "just a bunch of disks," referring to independent hard disks not in a RAID, or redundant array of independent disks, configuration.)

Spark is written in Scala, a language compiled to run on a Java virtual machine (JVM) with programming interfaces in Python (PySpark), Scala, and Java. The following are software prerequisites for installing and running Spark:

- Java (JDK preferably)
- Python, if you intend to use PySpark
- R, if you wish to use Spark with R; as discussed in Chapter 8, "Introduction to Data Science and Machine Learning Using Spark"
- Git, Maven, or SBT, which may be useful if you intend to build Spark from source or compile Spark programs

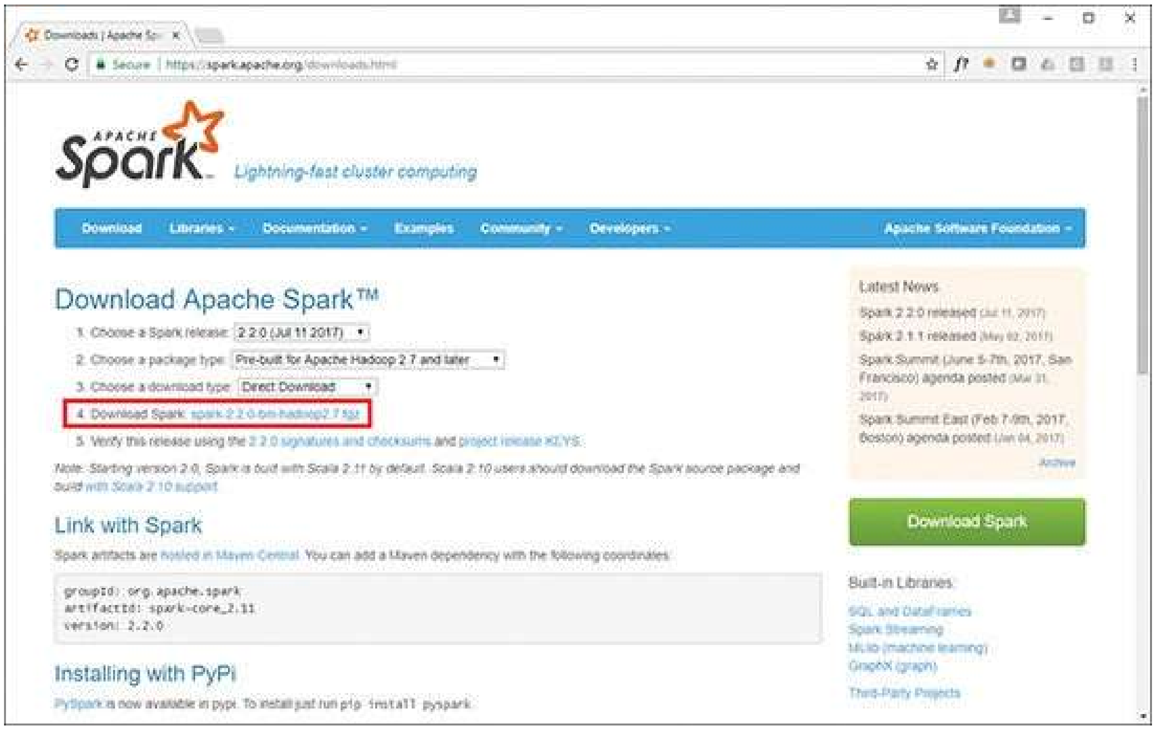
Getting Spark

Using a Spark release is often the easiest way to install Spark on a given system. Spark releases are downloadable from http://spark.apache.org/downloads.html.

These releases are cross-platform: They target a JVM environment, which is platform agnostic.

Using the build instructions provided on the official Spark website, you could also download the source code for Spark and build it yourself for your target platform. This method is more complicated however.

If you download a Spark release, you should select the builds with Hadoop, as shown in Figure 2.2. The “with Hadoop” Spark releases do not actually include Hadoop, as the name may imply. These releases simply include libraries to integrate with the Hadoop clusters and distributions listed. Many of the Hadoop classes are required, regardless of whether you are using Hadoop with Spark.


Downloads | Apache Spark
https://spark.apache.org/downloads.html
Lightning-fast cluster computing
Download
Libraries
Documentation
Examples
Community
Developers
Apache Software Foundation

Download Apache Spark™

1. Choose a Spark release: 2.2.0 (Jul 11 2017)
2. Choose a package type: Pre-built for Apache Hadoop 2.7 and later
3. Choose a download type: Direct Download
4. Download Spark: spark-2.2.0-bin-hadoop2.7.tgz
5. Verify this release using the 2.2.0 signatures and checksums and project release KEYS.

Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build with Scala 2.10 support.

Link with Spark

Spark artifacts are hosted in Maven Central. You can add a Maven dependency with the following coordinates:

```
groupId: org.apache.spark
artifactId: spark-core_2.11
version: 2.2.0
```

Installing with PyPi

PySpark is now available in pypi. To install just run `pip install pyspark`.

Latest News

Spark 2.2.0 released (Jul 11, 2017)

Spark 2.1.1 released (May 02, 2017)

Spark Summit (June 5-7th, 2017, San Francisco) agenda posted (Mar 31, 2017)

Spark Summit East (Feb 7-9th, 2017, Boston) agenda posted (Jan 04, 2017)

Archive

Download Spark

Built-in Libraries:

SQL and DataFrames
Spark Streaming
MLlib (machine learning)
GraphX (graph)

Third-Party Projects

Figure 2.2 Downloading a Spark release.

Using the “Without Hadoop” Builds

You may be tempted to download the “without Hadoop,” “user-provided Hadoop,” or “spark-x.x.x-bin-without-hadoop.tgz” options if you are installing in Standalone mode and not using Hadoop. The nomenclature can be confusing, but this build expects many of the required classes that are implemented in Hadoop to be present on the system. Generally speaking, you are usually better off downloading one of the spark-x.x.x-bin-hadoopx.x builds.

Spark is typically available with most commercial Hadoop distributions, including the following:

- Cloudera Distribution of Hadoop (CDH)
- Hortonworks Data Platform (HDP)
- MapR Converged Data Platform

In addition, Spark is available from major cloud providers through managed Hadoop offerings, including AWS EMR, Google Cloud Dataproc, and Microsoft Azure HDInsight.

If you have a Hadoop environment, you may have everything you need to get started and can skip the subsequent sections on installing Spark.

Installing Spark on Linux or Mac OS X

Linux is the most common and easiest platform to install Spark on, followed by Mac OS X. Installation on these two platforms is similar because they share the same kernel and have a similar shell environment. This exercise shows how to install Spark on an Ubuntu distribution of Linux; however, the steps are similar for installing Spark on another distribution of Linux or on Mac OS X (only using different package managers, such as, `yum`). Follow these steps to install Spark on Linux:

1. **Install Java.** It is general practice to install a JDK (Java Development Kit), which includes the Java Runtime Engine (JRE) and tools for building and managing Java or Scala applications. To do so, run the following:

Click here to view code image

```
$ sudo apt-get install openjdk-8-jdk-headless
```

Test the installation by running `java -version` in a terminal session; you should see output similar to the following if the installation is successful:

Click here to view code image

```
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-
2ubuntu1.17.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

On macOS, the java installation command is as follows:

Click here to view code image

```
$ brew cask install java
```

2. **Get Spark.** Download a release of Spark, using `wget` and the appropriate URL to download the release; you can obtain the actual download address from the http://spark.apache.org/downloads.html page shown in Figure 2.2. Although there is likely to be a later release available to you by the time you read this book, the following example shows a download of release 2.2.0:

Click here to view code image

```
$ wget https://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-
hadoop2.7.tgz
```

3. **Unpack the Spark release.** Unpack the Spark release and move it into a shared directory, such as `/opt/spark`:

Click here to view code image

```
$ tar -xzf spark-2.2.0-bin-hadoop2.7.tgz
$ sudo mv spark-2.2.0-bin-hadoop2.7 /opt/spark
```

4. **Set the necessary environment variables.** Set the `SPARK_HOME` variable and update the `PATH` variable as follows:

Click here to view code image

```
$ export SPARK_HOME=/opt/spark
$ export PATH=$SPARK_HOME/bin:$PATH
```

You may wish to set these on a persistent or permanent basis (for example, using `/etc/environment` on an Ubuntu instance).

5. **Test the installation.** Test the Spark installation by running the built-in Pi Estimator example in Local mode, as follows:

Click here to view code image

```
$ spark-submit --class org.apache.spark.examples.SparkPi \
--master local \
$SPARK_HOME/examples/jars/spark-examples*.jar 1000
```

If successful, you should see output similar to the following among a large amount of informational log messages (which you will learn how to minimize later in this chapter):

```
Pi is roughly 3.1414961114149613
```

You can test the interactive shells, `pyspark` and `spark-shell`, at the terminal prompt as well.

Congratulations! You have just successfully installed and tested Spark on Linux. How easy was that?

Installing Spark on Windows

Installing Spark on Windows can be more involved than installing Spark on Linux or Mac OS X because many of the dependencies, such as Python and Java, need to be addressed first. This example uses Windows Server 2012, the server version of Windows 8.1. You need a decompression utility capable of extracting `.tar.gz` and `.gz` archives because Windows does not have native support for these archives. 7-Zip, which you can obtain from http://7-zip.org/download.html, is a suitable program for this. When you have the needed compression utility, follow these steps:

1. **Install Python.** As mentioned earlier, Python is not included with Windows, so you need to download and install it. You can obtain a Windows installer for Python from https://www.python.org/getit/ or https://www.python.org/downloads/windows/. This example uses Python 2.7.10, so select `C:\Python27` as the target directory for the installation.
2. **Install Java.** In this example, you will download and install the latest Oracle JDK. You can obtain a Windows installer package from http://www.oracle.com/technetwork/java/javase/downloads/index.html. To confirm that Java has been installed correctly and is available in the system ta `PATH`, type `java -version` at the Windows command prompt; you should see the version installed returned.
3. **Download and unpack a Hadoop release.** Download the latest Hadoop release from http://hadoop.apache.org/releases.html. Unpack the Hadoop release (using 7-Zip or a similar decompression utility) into a local directory, such as `C:\Hadoop`.
4. **Install Hadoop binaries for Windows.** In order to run Spark on Windows, you need several Hadoop binaries compiled for Windows, including `hadoop.dll` and `winutils.exe`. The Windows-specific libraries and executables required for Hadoop are obtainable from https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-winutils. Download the `hadoop-winutils` archive and unpack the contents to the `bin` subdirectory of your Hadoop release (`C:\Hadoop\bin`).
5. **Download and unpack Spark.** Download the latest Spark release from

https://spark.apache.org/downloads.html, as shown in Figure 2.2. As discussed, use the "pre-built for Apache Hadoop" release corresponding to the Hadoop release downloaded in step 3. Unpack the Spark release into a local directory, such as `C:\Spark`.

6. **Disable IPv6.** Disable IPv6 for Java applications by running the following command as an administrator from the Windows command prompt:

Click here to view code image

```
C:\> setx _JAVA_OPTIONS "-Djava.net.preferIPv4Stack=true"
```

If you are using Windows PowerShell, you can enter the following equivalent command:

Click here to view code image

```
PS C:\>[Environment]::SetEnvironmentVariable("_JAVA_OPTIONS",
"-Djava.net.preferIPv4Stack=true", "User")
```

Note that you need to run these commands as a local administrator. For simplicity, this example shows applying all configuration settings at a *user* level. However, you can instead choose to apply any of the settings shown at a *machine* level—for instance, if you have multiple users on a system. Consult the documentation for Microsoft Windows for more information about this.

7. **Set the necessary environment variables.** Set the `HADOOP_HOME` environment variable by running the following command at the Windows command prompt:

Click here to view code image

```
C:\> setx HADOOP_HOME C:\Hadoop
```

Here is the equivalent using the Windows PowerShell prompt:

Click here to view code image

```
PS C:\>[Environment]::SetEnvironmentVariable("HADOOP_HOME",
"C:\Hadoop", "User")
```

8. **Set up the local metastore.** You need to create a location and set the appropriate permissions to a local metastore. We discuss the role of the metastore specifically in Chapter 6, "SQL and NoSQL Programming with Spark," when we begin to look at Spark SQL. For now, just run the following commands from the Windows or PowerShell command prompt:

Click here to view code image

```
C:\> mkdir C:\tmp\hive
C:\> Hadoop\bin\winutils.exe chmod 777 /tmp/hive
```

9. **Test the installation.** Open a Windows command prompt or PowerShell session and change directories to the `bin` directory of your Spark installation, as follows:

Click here to view code image

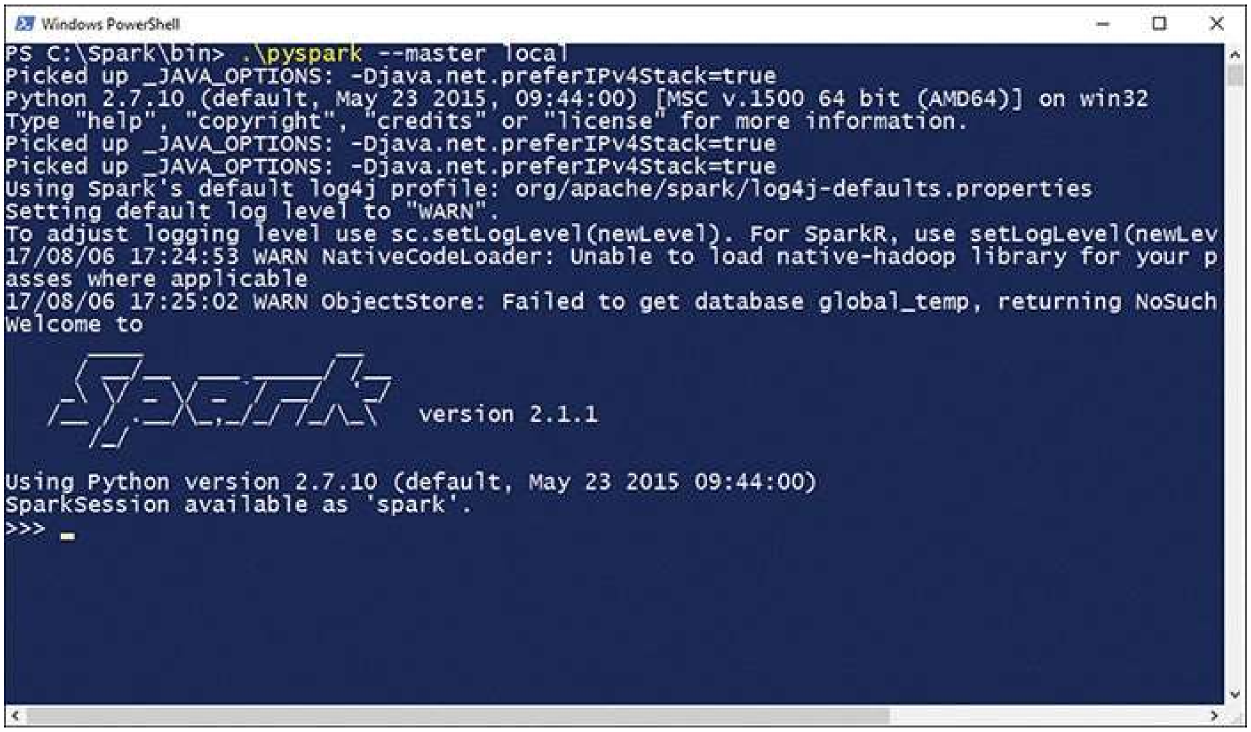
```
C:\> cd C:\Spark\bin
```

At the subsequent prompt, enter the `pyspark` command to open an interactive Python Spark shell:

Click here to view code image

```
C:\Spark\bin> pyspark --master local
```

Figure 2.3 shows an example of what you should expect to see using Windows PowerShell.



```
PS C:\Spark\bin> .\pyspark --master local
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Python 2.7.10 (default, May 23 2015, 09:44:00) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLev
17/08/06 17:24:53 WARN NativeCodeLoader: Unable to load native-hadoop library for your p
asses where applicable
17/08/06 17:25:02 WARN ObjectStore: Failed to get database global_temp, returning NoSuch
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Python version 2.7.10 (default, May 23 2015 09:44:00)
SparkSession available as 'spark'.
>>> 
```

Figure 2.3 PySpark in Windows PowerShell.

Enter `quit()` to exit the shell.

Now run the built-in Pi Estimator sample application by running the following from the command prompt:

Click here to view code image

```
C:\Spark\bin> spark-submit --class org.apache.spark.examples.SparkPi
 --master local C:\Spark\examples\jars\spark-examples*.jar 100
```

You now see a lot of informational logging messages; within these messages you should see something that resembles the following:

Click here to view code image

```
Pi is roughly 3.1413223141322315
```

Congratulations! You have just successfully installed and tested Spark on Windows.

Exploring the Spark Installation

It is worth getting familiar with the contents of the Spark installation directory, sometimes referred to as the `SPARK_HOME`. Table 2.1 provides an overview of the directories within the `SPARK_HOME`.

Table 2.1 **Spark Installation Contents**

Directory	**Description**
`bin/`	Contains all the commands/scripts to run Spark applications interactively through shell programs such as `pyspark`, `spark-shell`, `spark-sql`, and `sparkR`, or in batch mode using `spark-submit`.
`conf/`	Contains templates for Spark configuration files, which you can use to set Spark configuration values (`spark-defaults.conf.template`), as well as a shell script used to set environment variables required for Spark processes (`spark-env.sh.template`). There are also configuration templates to control logging (`log4j.properties.template`), a metrics collection (`metrics.properties.template`), and a template for the `slaves` file (`slaves.template`), which controls which slave nodes can join the Spark cluster running in Standalone mode.
`data/`	Contains sample datasets used for testing the `mllib`, `graphx`, and `streaming` libraries within the Spark project (all of which are discussed later in this book).
`examples/`	Contains the source code and compiled assemblies (jar files) for all the examples shipped with the Spark release, including the Pi

	Estimator application used in previous examples. Sample programs are included in Java, Python, R, and Scala. You can also find the latest code for the included examples at https://github.com/apache/spark/tree/master/examples.
`jars/`	Contains the main assemblies for Spark as well as assemblies for support services used by Spark, such as `snappy`, `py4j`, `parquet`, and more. This directory is included in the `CLASSPATH` for Spark by default.
`licenses/`	Includes license files covering other included projects, such as Scala and JQuery. These files are for legal compliance purposes only and are not required to run Spark.
`python/`	Contains all the Python libraries required to run PySpark. You generally don't need to access these files directly.
`R/`	Contains the `SparkR` package and associated libraries and documentation. You will learn about `SparkR` in Chapter 8, "Introduction to Data Science and Machine Learning Using Spark."
`sbin/`	Contains administrative scripts to start and stop master and slave services for Spark clusters running in Standalone mode, locally or remotely, as well as start processes related to YARN and Mesos. You will use some of these scripts in the next section when you deploy a multi-node cluster in Standalone mode.
`yarn/`	Contains support libraries for Spark applications running on YARN. This includes the shuffle service, a support service Spark uses to move data between processes in a YARN cluster.

The remainder of this book references many of the directories listed in Table 2.1.

Deploying a Multi-Node Spark Standalone Cluster

Now that you have installed and tested a Spark installation in Local mode, it's time to unleash the true power of Spark by creating a fully distributed Spark cluster. For this exercise, you will use four Linux hosts to create a simple three-node cluster using the Standalone scheduler. Follow these steps:

1. **Plan a cluster topology and install Spark on multiple systems.** Because

this is a distributed system, you need to install Spark, as shown in the previous exercises, on three additional hosts. In addition, you need to designate one host as the Spark Master and the other hosts as Workers. For this exercise, the first host is named *sparkmaster*, and the additional hosts are names *sparkworker1*, *sparkworker2*, and *sparkworker3*.

2. **Configure networking.** All nodes in the Spark cluster need to communicate will all other hosts in the cluster. The easiest way to accomplish this is by using `hosts` files (entries for all hosts in `/etc/hosts` on each system). Ensure that each node can resolve the other. The `ping` command can be used for this; for example, here is how you use it from the *sparkmaster* host:

Click here to view code image

```
$ ping sparkworker1
```

3. **Create and edit a `spark-defaults.conf` file on each host.** To create and configure a `spark-defaults.conf` file on each node, run the following commands on the *sparkmaster* and *sparkworker* hosts:

Click here to view code image

```
$ cd $SPARK_HOME/conf
$ sudo cp spark-defaults.conf.template spark-defaults.conf
$ sudo sed -i "\$aspark.master\tspark://sparkmaster:7077" spark-
defaults.conf
```

4. **Create and edit a `spark-env.sh` file on each host.** To create and configure a `spark-env.sh` file on each node, complete the following tasks on the *sparkmaster* and *sparkworker* hosts:

Click here to view code image

```
$ cd $SPARK_HOME/conf
$ sudo cp spark-env.sh.template spark-env.sh
$ sudo sed -i "\$aSPARK_MASTER_IP=sparkmaster" spark-env.sh
```

5. **Start the Spark Master.** On the *sparkmaster* host, run the following command:

Click here to view code image

```
$ sudo $SPARK_HOME/sbin/start-master.sh
```

Test the Spark Master process by viewing the Spark Master web UI at http://sparkmaster:8080/.

6. **Start the Spark Workers.** On each *sparkworker* node, run the following command:

Click here to view code image

```
$ sudo $SPARK_HOME/sbin/start-slave.sh spark://sparkmaster:7077
```

Check the Spark Worker UIs on http://sparkslave*N*:8081/.

7. **Test the multi-node cluster.** Run the built-in Pi Estimator example from the terminal of any node in the cluster:

Click here to view code image

```
$ spark-submit --class org.apache.spark.examples.SparkPi \
--master spark://sparkmaster:7077 \
--driver-memory 512m \
--executor-memory 512m \
--executor-cores 1 \
$SPARK_HOME/examples/jars/spark-examples*.jar 10000
```

You should see output similar to that from the previous exercises.

You could also enable passwordless SSH (Secure Shell) for the Spark Master to the Spark Workers. This is required to enable remote login for the slave daemon startup and shutdown actions.

Deploying Spark in the Cloud

The proliferation of public and private cloud technology, Software-as-a-Service (SaaS), Infrastructure-as-a-Service (IaaS), and Platform-as-a-Service (PaaS) have been game changers in terms of the way organizations deploy technology.

You can deploy Spark in the cloud to provide a fast, scalable, and elastic processing environment. Several methods are available to deploy Spark platforms, applications, and workloads in the cloud; the following sections explore them.

Amazon Web Services (AWS)

Amazon has spent years designing and building scalable infrastructure, platforms, services, and APIs to manage its vast business requirements. The majority of these services are exposed for public consumption (paid, of course!) through Amazon Web Services (AWS).

The AWS portfolio contains dozens of different services, from IaaS products such as Elastic Compute Cloud (EC2), storage services such as S3, and PaaS products such as Redshift. AWS compute resources can be provisioned on demand and paid for on an hourly basis. They are also available as “spot” instances, which use a market mechanism to offer lower usage costs by taking advantage of low-demand periods.

There are two primary methods for creating Spark clusters in AWS: EC2 and Elastic MapReduce (EMR). To use any of the AWS deployment options, you need a valid AWS account and API keys if you are using the AWS software development kit (SDK) or command line interface (CLI).

Spark on EC2

You can launch Spark clusters (or Hadoop clusters capable of running Spark) on EC2 instances in AWS. Typically this is done within a Virtual Private Cloud (VPC), which allows you to isolate cluster nodes from public networks. Deployment of Spark clusters on EC2 usually involves deployment of configuration management tools such as Ansible, Chef, Puppet, or AWS CloudFormation, which can automate deployment routines using an Infrastructure-as-Code (IaC) discipline.

In addition, there are several predeveloped Amazon Machine Images (AMIs) available in the AWS Marketplace; these have a pre-installed and configured release of Spark.

You can also create Spark clusters on containers by using the EC2 Container Service. There are numerous options to create these, from existing projects available in GitHub and elsewhere.

Spark on EMR

Elastic MapReduce (EMR) is Amazon’s Hadoop-as-a-Service platform. EMR clusters are essentially Hadoop clusters with a variety of configurable ecosystem projects, such as Hive, Pig, Presto, Zeppelin, and, of course, Spark.

You can provision EMR clusters 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 applications to install, including Spark.

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

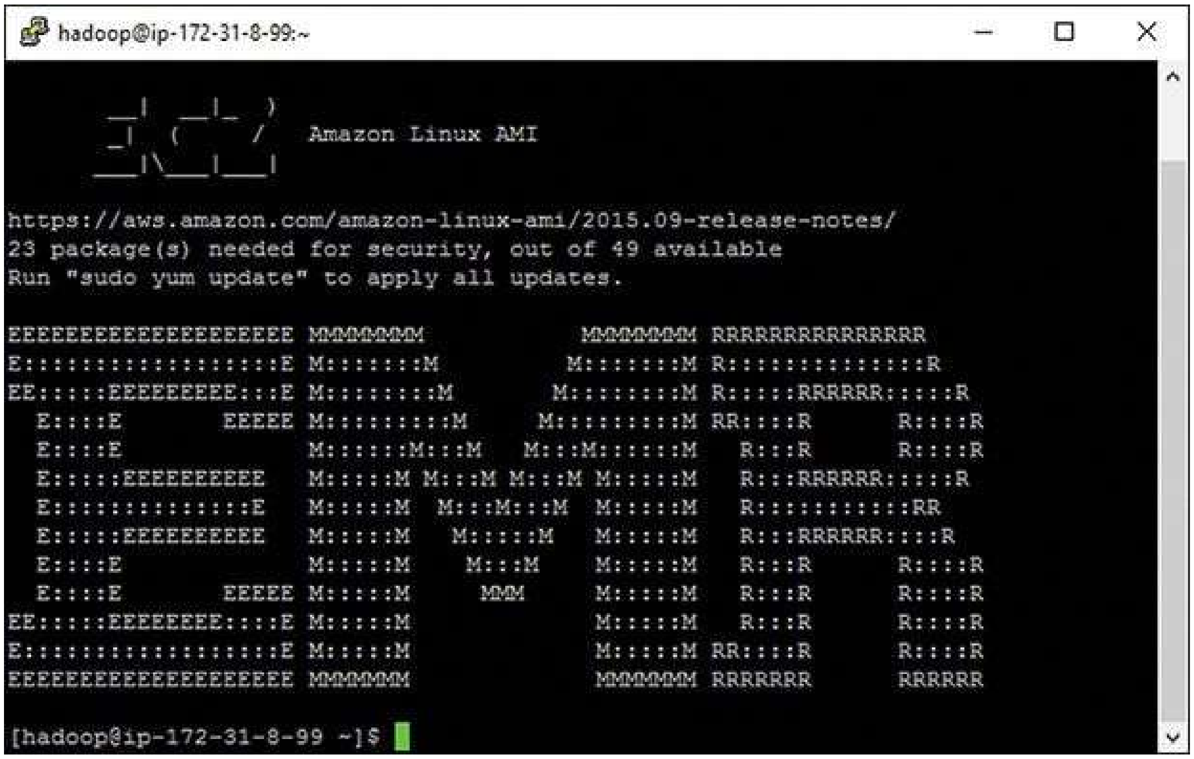
Listing 2.5 demonstrates creating a simple three-node EMR cluster with Spark and Zeppelin using the AWS CLI.

Listing 2.5 **Creating an EMR Cluster by Using the AWS CLI**

Click here to view code image

```
$ aws emr create-cluster \
--name "MyEMRCluster" \
--instance-type m1.xlarge \
--instance-count 3 \
--ec2-attributes KeyName=YOUR_KEY \
--use-default-roles \
--applications Name=Spark Name=Zeppelin-Sandbox
```

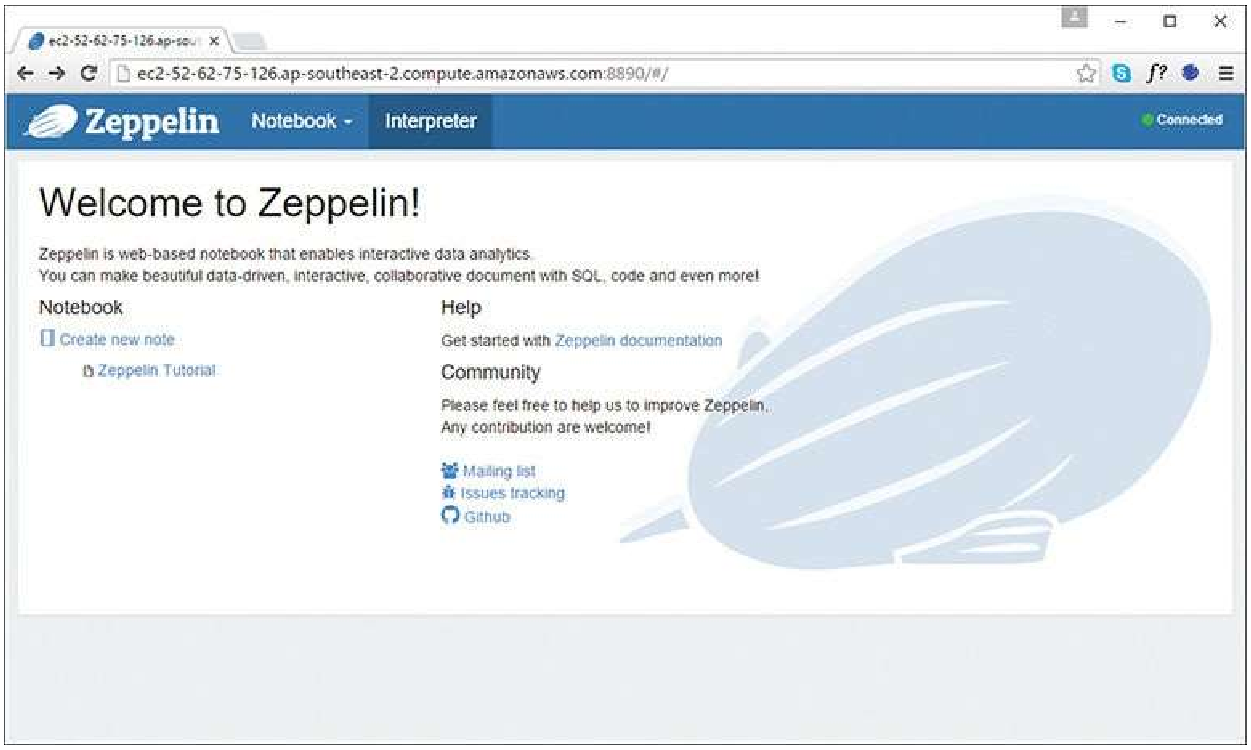
Figure 2.4 shows an EMR cluster console session.



```
hadoop@ip-172-31-8-99:~

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
23 package(s) needed for security, out of 49 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-8-99 ~]$
```

Figure 2.4 EMR console.

Figure 2.5 shows the Zeppelin notebook interface included with the EMR deployment, which can be used as a Spark programming environment.


ec2-52-62-75-126.ap-sou
ec2-52-62-75-126.ap-southeast-2.compute.amazonaws.com:8890/#/
Zeppelin
Notebook
Interpreter
Connected

Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Get started with Zeppelin documentation

Please feel free to help us to improve Zeppelin.
Any contribution are welcome!

Figure 2.5 Zeppelin interface.

Using EMR is a quick and scalable deployment method for Spark. For more information about EMR, go to https://aws.amazon.com/elasticmapreduce/.

Google Cloud Platform (GCP)

Much like Amazon does with AWS, Google deploys its vast array of global services, such as Gmail and Maps, using its cloud computing platform known as the Google Cloud Platform (GCP). Google's cloud offering supports most of the services available in AWS, but the company has many other offerings as well, including making available TPUs (Tensor Processing Units).

TensorFlow

TensorFlow is an open source software library that Google created specifically for training *neural networks*, an approach to *deep learning*. Neural networks are used to discover patterns, sequences, and relations in much the same way that the human brain does.

As with AWS, you could choose to deploy Spark using Google's IaaS offering, Compute, which requires you to deploy the underlying infrastructure. However, there is a managed Hadoop and Spark platform available with GCP called Cloud Dataproc, and it may be an easier option.

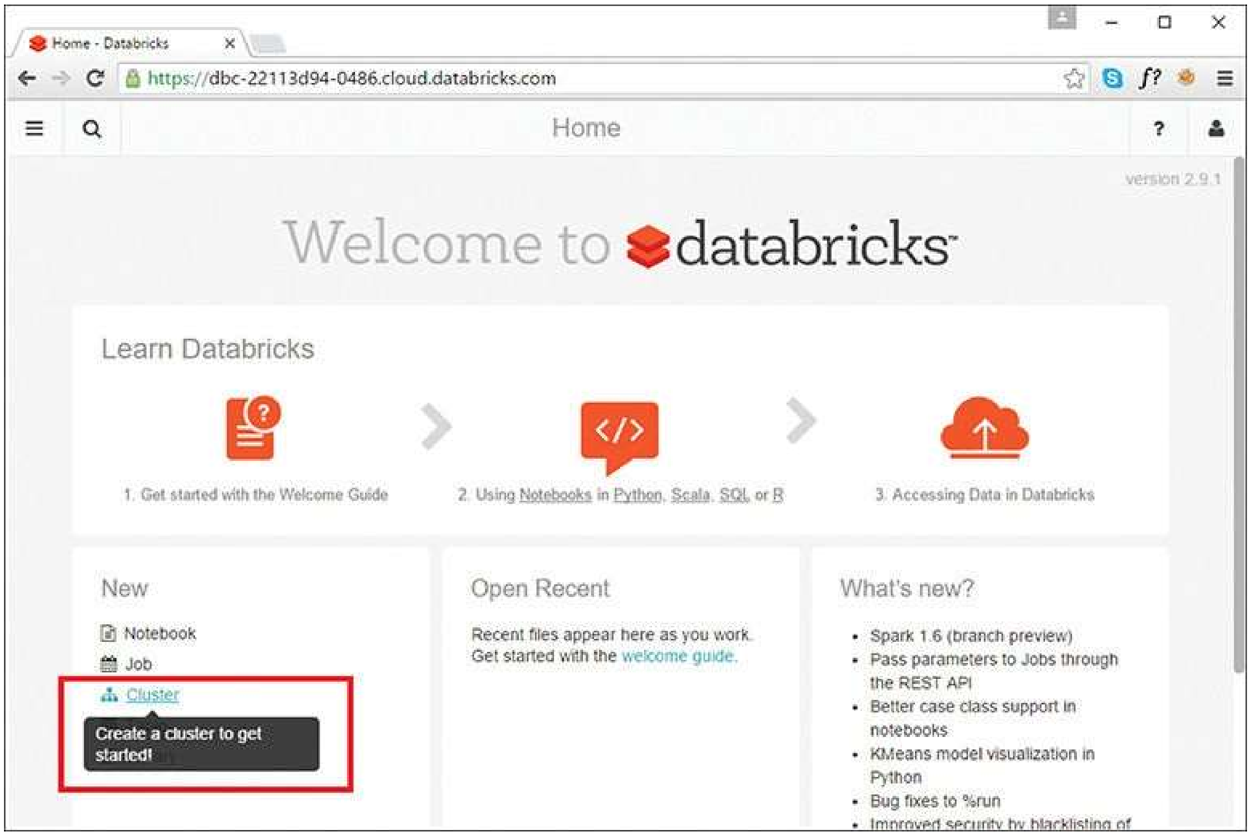
Cloud Dataproc offers a similarly managed software stack to AWS EMR, and you can deploy it to a cluster of nodes.

Databricks

Databricks is an integrated cloud-based Spark workspace that allows you to launch managed Spark clusters and ingest and interact with data from S3 or other relational database or flat-file data sources, either in the cloud or from your environment. The Databricks platform uses your AWS credentials to create its required infrastructure components, so you effectively have ownership of these assets in your AWS account. Databricks provides the deployment, management, and user/application interface framework for a cloud-based Spark platform in AWS.

Databricks has several pricing plans available, with different features spanning support levels, security and access control options, GitHub integration, and more. Pricing is subscription based, with a flat monthly fee plus nominal utilization charges (charged per hour per node). Databricks offers a 14-day free trial period to get started. You are responsible for the instance costs incurred in AWS for Spark clusters deployed using the Databricks platform; however, Databricks allows you to use discounted spot instances to minimize AWS costs. For the latest pricing and subscription information, go to https://databricks.com/product/pricing.

Databricks provides a simple deployment and user interface, shown in Figure 2.6, which abstracts the underlying infrastructure and security complexities involved in setting up a secure Spark environment in AWS. The Databricks management console allows you to create notebooks, similar to the Zeppelin notebook deployed with AWS EMR. There are APIs available from Databricks for deployment and management as well. These notebooks are automatically associated with your Spark cluster and provide seamless programmatic access to Spark functions using Python, Scala, SQL, or R.


Home - Databricks
https://dbc-22113d94-0486.cloud.databricks.com
Home
version 2.9.1
Welcome to databricks
Learn Databricks
1. Get started with the Welcome Guide
2. Using Notebooks in Python, Scala, SQL or R
3. Accessing Data in Databricks
New
Notebook
Job
Cluster
Open Recent
What's new?

Create a cluster to get started!

Recent files appear here as you work.
Get started with the welcome guide.

- Spark 1.6 (branch preview)
- Pass parameters to Jobs through the REST API
- Better case class support in notebooks
- KMeans model visualization in Python
- Bug fixes to %run
- Improved security by blacklisting of

Figure 2.6 Databricks console.

Databricks has its own distributed filesystem called the Databricks File System (DBFS). DBFS allows you to mount existing S3 buckets and make them seamlessly available in your Spark workspace. You can also cache data on the solid-state disks (SSDs) of your worker nodes to speed up access. The `dbutils` library included in your Spark workspace allows you to configure and interact with the DBFS.

The Databricks platform and management console allows you to create data objects as tables—which is conceptually similar to tables in a relational database—from a variety of sources, including AWS S3 buckets, Java Database Connectivity (JDBC) data sources, the DBFS, or by uploading your own files using drag-and-drop functionality. You can also create jobs by using the Databricks console, and you can run them non-interactively on a user-defined schedule.

The core AMP Labs team that created—and continues to be a major contributor to—the Spark project founded the Databricks company and platform. Spark releases and new features are typically available in the Databricks platform before they are shipped with other distributions, such as CDH or HDP. More information about Databricks is available at http://databricks.com.

Summary

In this chapter, you have learned how to install Spark and considered the various prerequisite requirements and dependencies. You have also learned about the various deployment modes available for deploying a Spark cluster, including Local, Spark Standalone, YARN, and Mesos. In the first exercise, you set up a fully functional Spark Standalone cluster. In this chapter you also looked at some of the cloud deployment options available for deploying Spark clusters, such as AWS EC2 or EMR clusters, Google Cloud Dataproc, and Databricks. Any of the deployments discussed or demonstrated in this chapter can be used for programming exercises throughout the remainder of this book—and beyond.