O'REILLY®

Compliments of pentaho®

PREVIEW EDITION

# Learning Spark

LIGHTNING-FAST DATA ANALYTICS

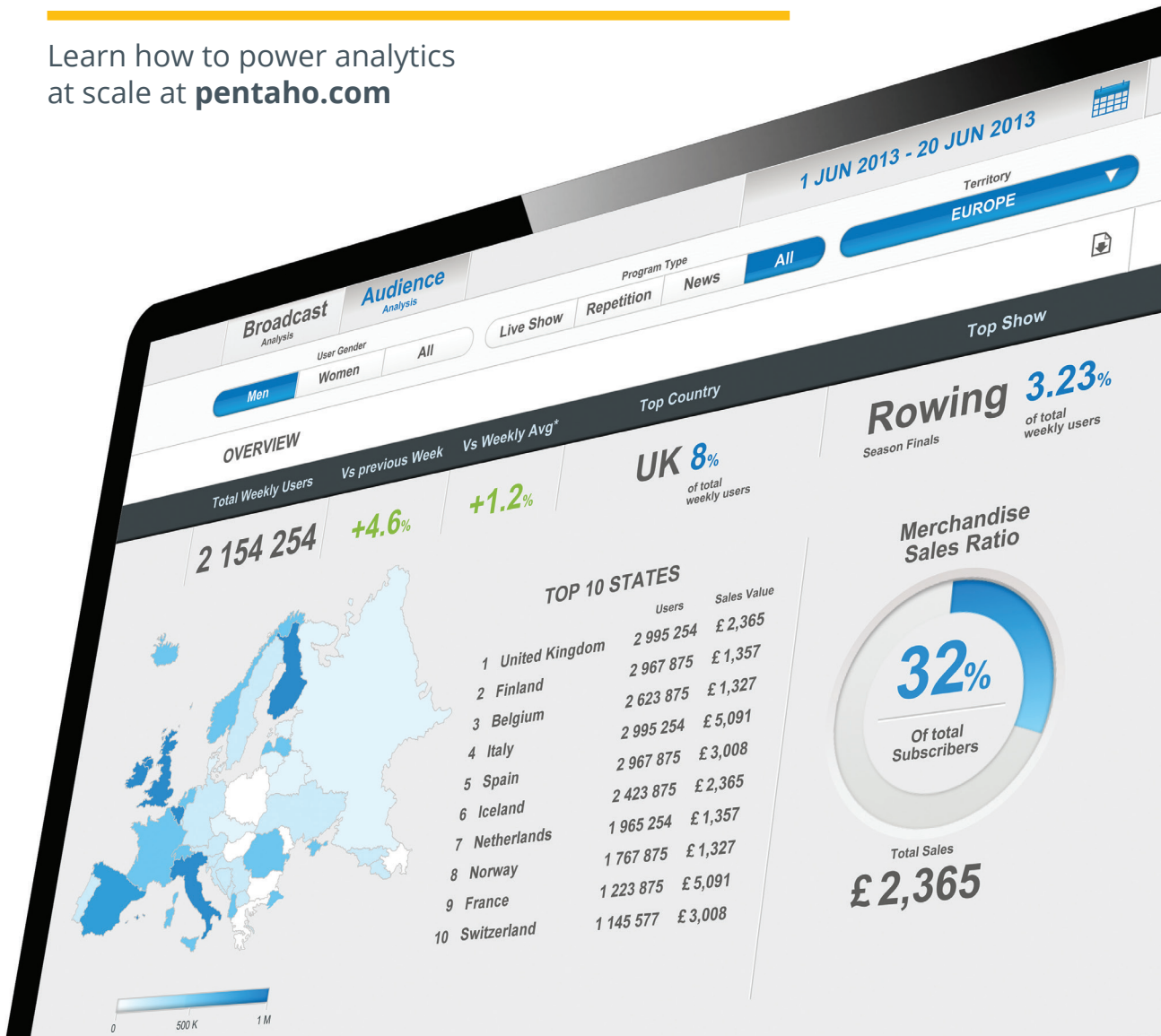
Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia

pentaho

# Bring Your **Big Data to Life**

## Big Data Integration and Analytics

Learn how to power analytics
at scale at **pentaho.com**



This Preview Edition of *Learning Spark*, *Chapter 1*, is a work in progress. The final book is currently scheduled for release in February 2015 and will be available at *oreilly.com* and other retailers once it is published.

# Learning Spark

*Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia*

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

**Learning Spark**

by Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia



Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

**Editors:** Ann Spencer and Marie Beaugureau
**Production Editor:** Kara Ebrahim
**Cover Designer:** Karen Montgomery
**Interior Designer:** David Futato
**Illustrator:** Rebecca Demarest

February 2015: First Edition

**Revision History for the First Edition:**

2014-11-25: Preview Edition

See *http://oreilly.com/catalog/errata.csp?isbn=9781449358624* for release details.



ISBN: 978-1-449-35862-4

[?]

# Table of Contents

# Preface

As parallel data analysis has become increasingly common, practitioners in many fields have sought easier tools for this task. Apache Spark has quickly emerged as one of the most popular tools for this purpose, extending and generalizing MapReduce. Spark offers three main benefits. First, it is easy to use — you can develop applications on your laptop, using a high-level API that lets you focus on the content of your computation. Second, Spark is fast, enabling interactive use and complex algorithms. And third, Spark is a *general* engine, allowing you to combine multiple types of computations (e.g., SQL queries, text processing and machine learning) that might previously have required learning different engines. These features make Spark an excellent starting point to learn about big data in general.

This introductory book is meant to get you up and running with Spark quickly. You'll learn how to download and run Spark on your laptop and use it interactively to learn the API. Once there, we'll cover the details of available operations and distributed execution. Finally, you'll get a tour of the higher-level libraries built-into Spark, including libraries for machine learning, stream processing, graph analytics and SQL. We hope that this book gives you the tools to quickly tackle data analysis problems, whether you do so on one machine or hundreds.

## Audience

This book targets Data Scientists and Engineers. We chose these two groups because they have the most to gain from using Spark to expand the scope of problems they can solve. Spark's rich collection of data focused libraries (like MLlib) make it easy for data scientists to go beyond problems that fit on single machine while making use of their statistical background. Engineers, meanwhile, will learn how to write general-purpose distributed programs in Spark and operate production applications. Engineers and data scientists will both learn different details from this book, but will both be able to apply Spark to solve large distributed problems in their respective fields.

Data scientists focus on answering questions or building models from data. They often have a statistical or math background and some familiarity with tools like Python, R and SQL. We have made sure to include Python, and where relevant SQL, examples for all our material, as well as an overview of the machine learning and advanced analytics libraries in Spark. If you are a data scientist, we hope that after reading this book you will be able to use the same mathematical approaches to solving problems, except much faster and on a much larger scale.

The second group this book targets is software engineers who have some experience with Java, Python or another programming language. If you are an engineer, we hope that this book will show you how to set up a Spark cluster, use the Spark shell, and write Spark applications to solve parallel processing problems. If you are familiar with Hadoop, you have a bit of a head start on figuring out how to interact with HDFS and how to manage a cluster, but either way, we will cover basic distributed execution concepts.

Regardless of whether you are a data analyst or engineer, to get the most of this book you should have some familiarity with one of Python, Java, Scala, or a similar language. We assume that you already have a solution for storing your data and we cover how to load and save data from many common ones, but not how to set them up. If you don't have experience with one of those languages, don't worry, there are excellent resources available to learn these. We call out some of the books available in Supporting Books.

## How This Book is Organized

The chapters of this book are laid out in such a way that you should be able to go through the material front to back. At the start of each chapter, we will mention which sections of the chapter we think are most relevant to data scientists and which sections we think are most relevant for engineers. That said, we hope that all the material is accessible to readers of either background.

The first two chapters will get you started with getting a basic Spark installation on your laptop and give you an idea of what you can accomplish with Apache Spark. Once we've got the motivation and setup out of the way, we will dive into the Spark Shell, a very useful tool for development and prototyping. Subsequent chapters then cover the Spark programming interface in detail, how applications execute on a cluster, and higher-level libraries available on Spark such as Spark SQL and MLlib.

## Supporting Books

If you are a data scientist and don't have much experience with Python, the Learning Python and Head First Python books are both excellent introductions. If you have some Python experience and want some more, Dive into Python is a great book to get a deeper understanding of Python.

If you are an engineer and after reading this book you would like to expand your data analysis skills, Machine Learning for Hackers and Doing Data Science are excellent books from O'Reilly.

This book is intended to be accessible to beginners. We do intend to release a deep dive follow-up for those looking to gain a more thorough understanding of Spark's internals.

## Code Examples

All of the code examples found in this book are on GitHub. You can examine them and check them out from *https://github.com/databricks/learning-spark*. Code examples are provided in Java, Scala, and Python.

Our Java examples are written to work with Java version 6 and higher. Java 8 introduces a new syntax called "lambdas" that makes writing inline functions much easier, which can simplify Spark code. We have chosen not to take advantage of this syntax in most of our examples, as most organizations are not yet using Java 8. If you would like to try Java 8 syntax, you can see the Databricks blog post on this topic. Some of the examples will also be ported to Java 8 and posted to the books GitHub.

## Early Release Status and Feedback

This is an **early release** copy of Learning Spark, and as such we are still working on the text, adding code examples, and writing some of the later chapters. Although we hope that the book is useful in its current form, we would greatly appreciate your feedback so we can improve it and make the best possible finished product. The authors and editors can be reached at *book-feedback@databricks.com*.

The authors would like to thank the reviewers who offered feedback so far: Joseph Bradley, Dave Bridgeland, Chaz Chandler, Mick Davies, Sam DeHority, Ali Ghodsi, Vida Ha, Juliet Hougland, Andrew Gal, Michael Gregson, Jan Joeppen, Stephan Jou, Jeff Martinez, Josh Mahonin, Mike Patterson, and Bruce Szalwinski.

The authors would also like to extend a special thanks Deborah Siegel, Dr. Normen Muller, and Sameer Farooqui. They provided detailed feedback on the majority of the chapters and helped point out many great improvements.

We would also like to thank the subject matter experts who took time to edit and write parts of their own chapters. Tathagata Das worked with us on a very tight schedule to get the Spark Streaming chapter finished. Tathagata went above and beyond with clarifying examples, answering many questions, and improving the flow of the text in addition to his technical contributions. Michael Armbrust helped us check the Spark SQL

chapter for correctness with an extremely short time line. Joseph Bradley provided an introductory example for MLlib in all of its APIs. Reza Zadeh provided text and code examples for dimensionality reduction. Xiangrui Meng, Joseph Bradley and Reza Zadeh also provided editing and technical feedback to improve the MLlib chapter.

CHAPTER 1

# Introduction to Data Analysis with Spark

This chapter provides a high level overview of what Apache Spark is. If you are already familiar with Apache Spark and its components, feel free to jump ahead to (to come).

## What is Apache Spark?

Apache Spark is a cluster computing platform designed to be *fast* and *general-purpose*.

On the speed side, Spark extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. Speed is important in processing large datasets as it means the difference between exploring data interactively and waiting minutes between queries, or waiting hours to run your program versus minutes. One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also faster than MapReduce for complex applications running on disk.

On the generality side, Spark is designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries and streaming. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to *combine* different processing types, which is often necessary in production data analysis pipelines. In addition, it reduces the management burden of maintaining separate tools.

Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala and SQL, and rich built-in libraries. It also integrates closely with other big data tools. In particular, Spark can run in Hadoop clusters and access any Hadoop data source, including Cassandra.

## A Unified Stack

The Spark project contains multiple closely-integrated components. At its core, Spark is a "computational engine" that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a *computing cluster*. Because the core engine of Spark is both fast and general-purpose, it powers multiple higher-level components specialized for various workloads, such as SQL or machine learning. These components are designed to interoperate closely, letting you combine them like libraries in a software project.

A philosophy of tight integration has several benefits. First, all libraries and higher level components in the stack benefit from improvements at the lower layers. For example, when Spark's core engine adds an optimization, SQL and machine learning libraries automatically speed up as well. Second, the costs associated with running the stack are minimized, because instead of running 5-10 independent software systems, an organization only needs to run one. These costs include deployment, maintenance, testing, support, and more. This also means that each time a new component is added to the Spark stack, every organization that uses Spark will immediately be able to try this new component. This changes the cost of trying out a new type of data analysis from downloading, deploying, and learning a new software project to upgrading Spark.

Finally, one of the largest advantages of tight integration is the ability to build applications that seamlessly combine different processing models. For example, in Spark you can write one application that uses machine learning to classify data in real time as it is ingested from streaming sources. Simultaneously analysts can query the resulting data, also in real-time, via SQL, e.g. to join the data with unstructured log files. In addition, more sophisticated data engineers & data scientists can access the same data via the Python shell for ad-hoc analysis. Others might access the data in standalone batch applications. All the while, the IT team only has to maintain one software stack.

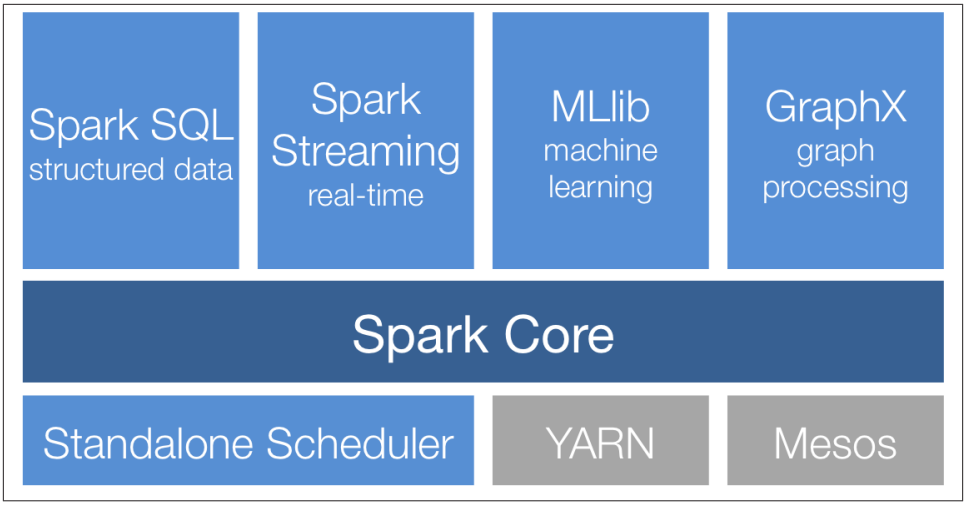



*Figure 1. The Spark Stack*

Here we will briefly introduce each of the components shown in Figure 1.

## Spark Core

Spark Core contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines Resilient Distributed Datasets (RDDs), which are Spark's main programming abstraction. RDDs represent a collection of items distributed across many compute nodes that can be manipulated in parallel. Spark Core provides many APIs for building and manipulating these collections.

## Spark SQL

Spark SQL is Spark's package for working with structured data. It allows querying data via SQL as well as the Apache Hive variant of SQL, called the Hive Query Language (HQL), and it supports many sources of data including Hive tables, Parquet, and JSON. Beyond providing a SQL interface to Spark, Spark SQL allows developers to intermix SQL queries with the programmatic data manipulations supported by RDDs in Python, Java and Scala, all within a single application, thus combining SQL with complex analytics. This tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open source data warehouse tool. Spark SQL was added to Spark in version 1.0.

Shark was an older SQL-on-Spark project out of UC Berkeley that modified Apache Hive to run on Spark. It has now been replaced by Spark SQL to provide better intergration with the Spark engine and language APIs.

## Spark Streaming

Spark Streaming is a Spark component that enables processing live streams of data. Examples of data streams include log files generated by production web servers, or queues of messages containing status updates posted by users of a web service. Spark Streaming provides an API for manipulating data streams that closely matches the Spark Core's RDD API, making it easy for programmers to learn the project and move between applications that manipulate data stored in memory, on disk, or arriving in real-time. Underneath its API, Spark Streaming was designed to provide the same degree of fault tolerance, throughput, and scalability that the Spark Core provides.

## MLlib

Spark comes with a library containing common machine learning (ML) functionality called MLlib. MLlib provides multiple types of machine learning algorithms, including classification, regression, clustering and collaborative filtering, as well as supporting functionality such as model evaluation and data import. It also provides some lower level ML primitives including a generic gradient descent optimization algorithm. All of these methods are designed to scale out across a cluster.

## GraphX

GraphX is a library that provides an API for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations. Like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge. GraphX also provides set of operators for manipulating graphs (e.g., `subgraph` and `mapVertices`) and a library of common graph algorithms (e.g., PageRank and triangle counting).

## Cluster Managers

Cluster Managers are a bit different as the previous components are things that are built on Spark, but Spark can run on different cluster managers. Under the hood, Spark is designed to efficiently scale up from one to many thousands of compute nodes. To achieve this while maximizing flexibility, Spark can run over a variety of *cluster managers*, including Hadoop YARN, Apache Mesos, and a simple cluster manager included in Spark itself called the Standalone Scheduler. If you are just installing Spark on an empty set of machines, the Standalone Scheduler provides an easy way to get started; while if you already have a Hadoop YARN or Mesos cluster, Spark's support for these allows your applications to also run on them. The (to come) explores the different options and how to choose the correct cluster manager.

# Who Uses Spark, and For What?

Because Spark is a general purpose framework for cluster computing, it is used for a diverse range of applications. In the Preface we outlined two personas that this book targets as readers: Data Scientists and Engineers. Let's take a closer look at each of these personas and how they use Spark. Unsurprisingly, the typical use cases differ across the two personas, but we can roughly classify them into two categories, *data science* and *data applications*.

Of course, these are imprecise personas and usage patterns, and many folks have skills from both, sometimes playing the role of the investigating Data Scientist, and then "changing hats" and writing a hardened data processing system. Nonetheless, it can be illuminating to consider the two personas and their respective use cases separately.

## Data Science Tasks

Data Science is the name of a discipline that has been emerging over the past few years centered around analyzing data. While there is no standard definition, for our purposes a *Data Scientist* is somebody whose main task is to analyze and model data. Data scientists may have experience using SQL, statistics, predictive modeling (machine learning), and some programming, usually in Python, Matlab or R. Data scientists also have experience with techniques necessary to transform data into formats that can be analyzed for insights (sometimes referred to as *data wrangling*).

Data Scientists use their skills to analyze data with the goal of answering a question or discovering insights. Oftentimes, their workflow involves ad-hoc analysis, and so they use interactive shells (vs. building complex applications) that let them see results of queries and snippets of code in the least amount of time. Spark's speed and simple APIs shine for this purpose, and its built-in libraries mean that many algorithms are available out of the box.

Spark supports the different tasks of data science with a number of components. The PySpark shell makes it easy to do interactive data analysis using Python. Spark SQL also has a separate SQL shell which can be used to do data exploration using SQL, or Spark SQL can be used as part of a regular Spark program or in the PySpark shell. Machine learning and data analysis is supported through the MLLib libraries. In addition support exists for calling out to existing programs in Matlab or R. Spark enables Data Scientists to tackle problems with larger data sizes than they could before with tools like R or Pandas.

Sometimes, after the initial exploration phase, the work of a Data Scientist will be "productionized", or extended, hardened (i.e. made fault tolerant), and tuned to become a production data processing application, which itself is a component of a business application. For example, the initial investigation of a Data Scientist might lead to the creation of a production recommender system that is integrated into a web application

and used to generate customized product suggestions to users. Often it is a different person or team that leads the process of productizing the work of the Data Scientists, and that person is often an Engineer.

### Data Processing Applications

The other main use case of Spark can be described in the context of the *Engineer* persona. For our purposes here, we think of Engineers as large class of software developers who use Spark to build production data processing applications. These developers usually have an understanding of the principles of software engineering, such as encapsulation, interface design, and Object Oriented Programming. They frequently have a degree in Computer Science. They use their engineering skills to design and build software systems that implement a business use case.

For Engineers, Spark provides a simple way to parallelize these applications across clusters, and hides the complexity of distributed systems programming, network communication and fault tolerance. The system gives enough control to monitor, inspect and tune applications while allowing common tasks to be implemented quickly. The modular nature of the API (based on passing distributed collections of objects) makes it easy to factor work into reusable libraries and test it locally.

Spark's users choose to use it for their data processing applications because it provides a wide variety of functionality, is easy to learn and use, and is mature and reliable.

## A Brief History of Spark

Spark is an open source project that has been built and is maintained by a thriving and diverse community of developers from many different organizations. If you or your organization are trying Spark for the first time, you might be interested in the history of the project. Spark started in 2009 as a research project in the UC Berkeley RAD Lab, later to become the AMPLab. The researchers in the lab had previously been working on Hadoop MapReduce, and observed that MapReduce was inefficient for iterative and interactive computing jobs. Thus, from the beginning, Spark was designed to be fast for interactive queries and iterative algorithms, bringing in ideas like support for in-memory storage and efficient fault recovery.

Research papers were published about Spark at academic conferences and soon after its creation in 2009, it was already 10—20x faster than MapReduce for certain jobs.

Some of Spark's first users were other groups inside of UC Berkeley, including machine learning researchers such as the the Mobile Millennium project, which used Spark to monitor and predict traffic congestion in the San Francisco bay Area. In a very short time, however, many external organizations began using Spark, and today, over 50 or-

ganizations list themselves on the Spark PoweredBy page [1], and dozens speak about their use cases at Spark community events such as Spark Meetups [2] and the Spark Summit [3]. Apart from UC Berkeley, major contributors to the project currently include Databricks, Yahoo! and Intel.

In 2011, the AMPLab started to develop higher-level components on Spark, such as Shark (Hive on Spark)[4] and Spark Streaming. These and other components are sometimes referred to as the Berkeley Data Analytics Stack (BDAS) [5].

Spark was first open sourced in March 2010, and was transferred to the Apache Software Foundation in June 2013, where it is now a top-level project.

## Spark Versions and Releases

Since its creation Spark has been a very active project and community, with the number of contributors growing with each release. Spark 1.0 had over 100 individual contributors. Though the level of activity has rapidly grown, the community continues to release updated versions of Spark on a regular schedule. Spark 1.0 was released in May 2014. This book focuses primarily on Spark 1.1.0 and beyond, though most of the concepts and examples also work in earlier versions.

## Persistence layers for Spark

Spark can create distributed datasets from any file stored in the Hadoop distributed file system (HDFS) or other storage systems supported by the Hadoop APIs (including your local file system, Amazon S3, Cassandra, Hive, HBase, etc). Its important to remember that Spark does not require Hadoop, it simply has support for storage systems implementing the Hadoop APIs. Spark supports text files, SequenceFiles, Avro, Parquet, and any other Hadoop InputFormat. We will look at interacting with these data sources in the loading and saving chapter.

1. *https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark*
2. *http://www.meetup.com/spark-users/*
3. *http://spark-summit.org*
4. Shark has been replaced by Spark SQL
5. *https://amplab.cs.berkeley.edu/software*