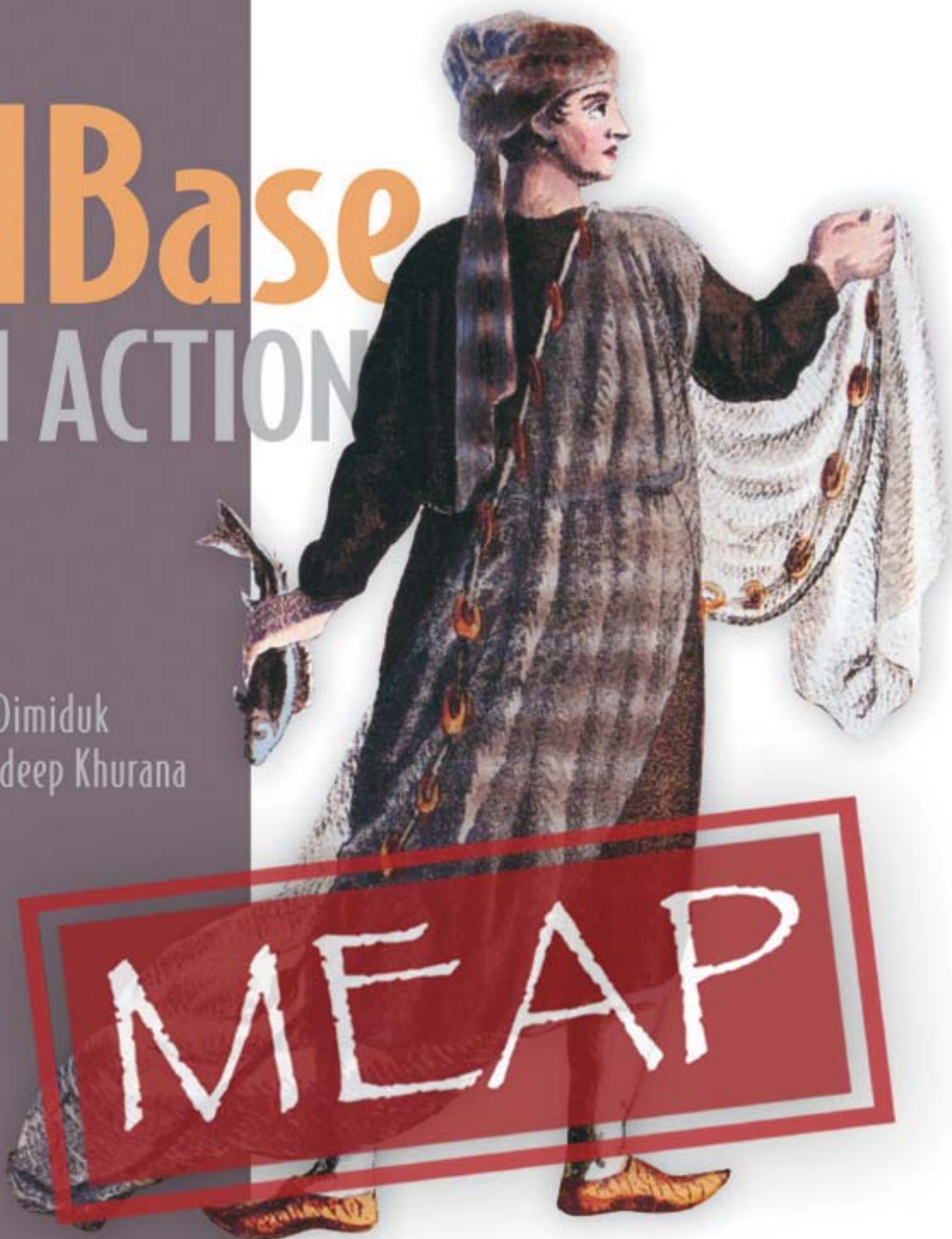


# HBase

IN ACTION

Nick Dimiduk  
Amandeep Khurana



MEAP



**MEAP Edition  
Manning Early Access Program  
HBase in Action version 2**

Copyright 2012 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

# *Table of Contents*

1. Introducing HBase

## **Part 1: Foundations**

2. Getting started

3. Hadoop foundations

4. Distributed, reliable, available

5. Schema design

## **Part 2: Adjacent concerns**

6. Coprocessors

7. Beyond Java

## **Part 3: HBase in practice**

8. Example application: OpenTSDB

9. Example schemas: MD-HBase

10. From relational to HBase: Northwind

## **Part 4: HBase in production**

11. Deployment

12. Operations

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=802>

## **Appendixes**

- A. Installing Hadoop
- B. Beyond MapReduce
- C. HBase configuration parameters
- D. HBase shell commands
- E. HBase data utilities

# 1

## *Introducing HBase*

This chapter covers

- The origins of Hadoop, HBase, and NoSQL
- Common use cases for HBase
- A basic HBase installation
- Storing and querying data with HBase

HBase is a database, the **Hadoop database**. It is a tool for storing and retrieving data with random access, meaning you can write data as you like and read it back again as you need it. HBase stores structured and semi-structured data naturally so you can load it up with tweets and parsed log files and a catalog of all your products right along with their customer reviews. It can store unstructured data too, so long as it's not too large. Fill it with images if you'd like and short videos too, but keep your blue-ray collection and hard-drive snapshots elsewhere<sup>1</sup>.

HBase is not a relational database like the ones with which you're likely accustomed. It does not speak SQL or enforce relationships within your data. It will not guarantee transactions and it doesn't mind storing an integer in one row while a string in another for the same column. It also doesn't demand custom, expensive disk configurations or gobs of RAM all jammed into a single machine. In fact, HBase would much prefer you didn't.

HBase is a database designed to run on a cluster of computers instead of a single one, 10's to 100's of machines. Each node in the cluster provides a bit of storage, a bit of cache, and a bit of computation as well. You can even run it on a myriad of different hardware configurations within the same cluster, HBase won't mind. This also makes HBase incredibly

---

<sup>1</sup> As you'll see in Chapter 3, these larger files are a great use-case for the HDFS.

flexible and forgiving. No node is unique<sup>2</sup>, so if one of those machines breaks down, you just replace it with another – no need for an exact replica. This adds up to a powerful, robust approach to data that, until now, has not been commonly available to mere mortals.

### Join the Community

Unfortunately, the largest HBase cluster is not publically known. This kind of information easily falls under the realm of “business confidential” and is not often shared. For now the curious must rely on footnotes in publications, bullets in presentations, and the friendly, unofficial chatter you’ll find at user groups, meet-ups, and conferences.

So participate! It’s good for you and it’s how we became involved as well. HBase is an open source project in an extremely specialized space. It has well-financed competition from some of the largest software companies on the planet. It is the community that created HBase and the community that will keeps it competitive and innovative. Plus, they’re an intelligent, friendly, attractive group. Step up, say hello, and tell them we sent you!

Now that you understand a little more of what HBase is and what it isn’t, lets go deeper. We’ll begin with some context, an examination of why so many people, seemingly all at once, decided their relational databases no longer cut it. We’ll follow that up with a brief history of how HBase came about. After that you’ll see how HBase is being used to solve all manner of problems. If you are like us, you want to play with HBase before going much further. We’ll wrap up by installing HBase on your laptop, tossing in some data, and pulling it out again. Context is important, so let’s start at the beginning.

## 1.1 Hello Big Data

To understand HBase you need first to understand the term “Big Data”. To be honest, it’s become something of a loaded term, especially now that the enterprise marketing engines have gotten hold of it. We’ll keep this discussion as grounded as possible<sup>3</sup>.

What is Big Data? There’s an excellent video interview with Roger Magoulas, Director of Research at O’Reilly Media, answering just this question. “Big Data,” Magoulas says, “is when the data is large enough that you have to think about it... in order to gain some benefit from it.”<sup>4</sup> This is a loose definition and it shifts with changing technologies and their adoption, but is accurate nonetheless. Said another way, you know your data is “big” when the “default” technology choice for data of this kind doesn’t work. For example, your user data becomes “big” when it drags your database to a halt. Your webserver log data is “big” when that script for counting daily page views takes longer than 24 hours to process. Your laboratory sensor

---

<sup>2</sup> No node is unique with perhaps one exception, depending on how you look at it. See Chapter 3 for details.

<sup>3</sup> We could have made a “cloud” reference there but we didn’t. Be grateful.

<sup>4</sup> This is a great four-part series titled “Big Data: Technologies & Techniques for Large-Scale Data”. Check it out. <http://radar.oreilly.com/2009/03/big-data-technologies-report.html>.

data becomes “big” when the standard pattern detection algorithm runs out of memory. In all of these examples the individual records within the data are small but the corpus in total has become so vast as to become “big.”

Number of records is only one dimension on which data can become “big.” The other common dimension is individual record size. The genome of a single human weighs in at a whopping 500 gigabytes of compressed data<sup>5</sup> and that’s just a single individual and a single species. As genomic sequencing becomes less and less expensive, it’s not far-fetched to imagine a single research study drawing conclusions from the data of thousands of individuals. Big Data is big!

To make a sweeping generalization, the data management technologies we’re used to were not built with the Internet in mind. This network of interconnected people and devices has produced, among other things, an explosion of data far beyond anything the previous generation of technologists imagined. The result is an entire class of datasets for which no technical solution existed. Necessity, it is often said, is the mother of invention. This necessity fuelled the innovation that introduced HBase to the world. Next you’ll see the source of this innovation and how it leads to HBase. For those uninterested in a brief history lesson, skip ahead to section 1.2.

### **1.1.1 Data innovation**

On the forefront of this explosion of data were many prominent Internet companies, most notably Google, Amazon, Yahoo!, and Microsoft. Some of them generated their own data, others collected what was freely available, but in all cases, managing these vastly different kinds of datasets became core to doing business. They all started out by building on the technology available at the time but in all cases the limitations of this technology became limitations on the continued growth and success of the business. While data management technology wasn’t core to the business, it became essential for doing business. The ensuing internal investment in technical research resulted in many new experiments in data technology.

While many companies kept their research closely guarded, Google chose to talk about its successes. The publications that really shook things up were the Google File System and MapReduce papers.<sup>6</sup> Taken together, these papers represent a novel approach to the storage and processing of data. Shortly thereafter, Google published the Bigtable paper,<sup>7</sup> providing a complement to the storage paradigm provided by its File System. Other companies built on this momentum, both the ideas and the habit of publishing their successful experiments. As Google’s publications provided insight into indexing the Internet, Amazon published Dynamo<sup>8</sup>, demystifying a fundamental component of their shopping cart.

---

<sup>5</sup> Citation needed.

<sup>6</sup> For a more complete exploration of the concepts presented in these papers and their references, see Chapter 3.

<sup>7</sup> Likewise, concepts from the Bigtable paper are explored in Chapter 4.

<sup>8</sup> Dynamo: Amazon’s Highly Available Key-value Store – DeCandia, Hastorun, et al. 2007.

It didn't take long for all these new ideas to begin condensing into implementations. In the years following, the data management space has come to host all manner of projects. Some focus on fast key-value stores, others provide native data-structures or document-based abstractions. Equally diverse are the intended access patterns and data volumes these technologies support. Some forego writing data to disk, sacrificing immediate persistence for performance. Not even traditional ACID<sup>9</sup> guarantees are held sacred in this new breed of data management systems. While proprietary products do exist, the vast majority of these technologies are open source projects. About the only thing they can agree upon is that the relational model is not necessary and even not desirable for all data applications. Thus, these technologies as a collection have come to be known as "NoSQL".

So where does HBase fit into all this? We're glad you asked. HBase does qualify as a NoSQL store. It provides a **key-value API**, though with a twist not common in other key-value stores. It promises **strict consistency** of both reads and writes so clients can see data immediately after it is written. HBase runs on **multiple nodes** in a cluster instead of on a single machine. It does not expose this detail to its clients. Your application code doesn't know if it is talking to 1 node or 100, which makes things simpler for everyone. HBase is designed for **gigabytes to terabytes** of data so it optimizes for this use-case. It is a part of the **Hadoop ecosystem** and depends on some key features, like data redundancy and batch processing, to be provided by other parts of Hadoop.

Now that you have some context for the environment at large, let's consider specifically the beginnings of HBase.

### 1.1.2 The rise of HBase

Pretend for a minute you're working on an open source project for searching the web by crawling websites and indexing them. You have an implementation that works on a small cluster of machines but requires a lot of manual steps. Pretend further you're working on this project around the same time Google publishes papers of its data storage and processing frameworks. Clearly, you would jump on these publications and spearhead an open source implementation based on these works. Okay, maybe you didn't and I surely didn't, but not so for Doug Cutting and Mike Cafarella.

Built out of Apache Lucene, Nutch was their open source web-search project and the motivation for the first implementation of Hadoop.<sup>10</sup> From there, it began to receive lots of attention from Yahoo!, which hired Cutting and others to work on it full-time. From there, Hadoop was extracted out of Nutch and eventually became an Apache top-level project. With Hadoop well underway and the Bigtable paper published, the groundwork existed to implement an open source Bigtable on top of Hadoop. In 2007, Cafarella released code for an experimental, open source Bigtable. He called it HBase. The startup PowerSet decided to

---

<sup>9</sup> For those who don't know (or don't remember), ACID is an acronym standing for *atomicity*, *consistency*, *isolation*, and *durability*. These are fundamental principals used to reason about data systems. See <http://en.wikipedia.org/wiki/ACID> for an introduction.

<sup>10</sup> A short historical summary was published by Cutting at <http://blog.lucene.com/2009/08/10/joining-cloudera/>

dedicate Jim Kellerman and Michael Stack to work on this Bigtable analog as a way of contributing back to the open source community upon which it relied.<sup>11</sup>

HBase proved to be a powerful tool, especially in places where Hadoop was already in use. Even in its infancy, it quickly found production deployment and developer support from other companies. Today, HBase is a top-level Apache project with thriving developer and user communities. It has become the data backbone for production deployments worldwide, in companies like StumbleUpon, Trend Micro, Facebook, Twitter, and Adobe.

Next, let's have a look at how HBase is used today. Through this you'll gain a feel for what kinds of data problems HBase has been used to tackle.

## **1.2 HBase success stories**

Sometimes the best way to understand a software product is to look at how it is used. The kinds of problems it solves and how that solution fits into a larger application architecture can tell you a lot about a product. Because HBase has seen a number of publicized production deployments, we can do just that. This section elaborates on some of these more common applications, notably web search, social networking, and log processing. HBase is modeled after Google's Bigtable so we'll start our exploration with the canonical Bigtable problem: storing the Internet.

### **1.2.1 Web search**

Search is the act locating a known value. When you want to search for terms in an entire document corpus, it requires a two-step process. First, you generate an index over all those documents, associating each term in the corpus back to the documents in which it appears. Once you have an index, it's easy to look up the set of all documents that contain your terms. This is precisely what Google and the rest of the search engines of the day wanted to do. Their document corpus is the entire Internet; the search terms are whatever you type into the search box.

Bigtable, and by extension HBase, provide storage for this corpus of documents. Bigtable supports row-level access so crawlers can insert and update documents individually. The search index can be generated efficiently via MapReduce directly against Bigtable. Finally, individual document results can be retrieved directly. Support for all of these access patterns was key in influencing the design of Bigtable. Figure 1.1 illustrates the critical role of Bigtable in the web-search application

---

<sup>11</sup> See Kellerman's blog post at [http://www.bing.com/community/site\\_blogs/b/powerset/archive/2007/10/16/first-release-of-hbase-in-hadoop.aspx](http://www.bing.com/community/site_blogs/b/powerset/archive/2007/10/16/first-release-of-hbase-in-hadoop.aspx)

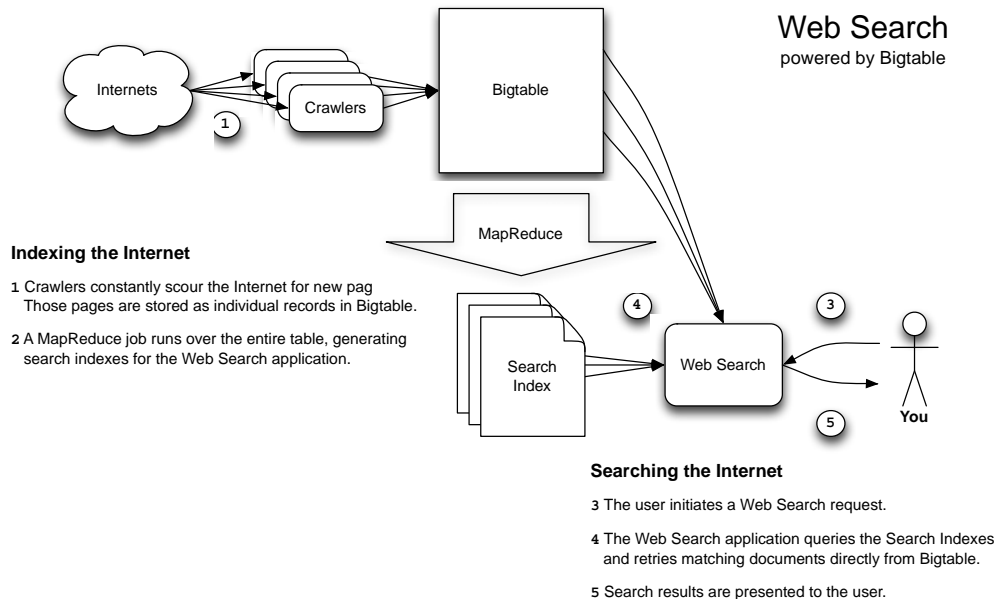


Figure 1.1: Providing web-search results using Bigtable, simplified. The crawlers, applications collecting web pages, store their data in Bigtable. A MapReduce process scans the table to produce the search index. Search results are queried from Bigtable to display to the user.

## NOTE

In the interest of brevity, this look at Bigtable doesn't do the original authors justice. We highly recommend all three papers, Google File System<sup>12</sup>, MapReduce<sup>13</sup>, and Bigtable<sup>14</sup> as required reading for anyone curious about these technologies. You won't be disappointed.

With the canonical HBase example covered, let's look at other places where HBase has found purchase. We'll start with the Web2.0 poster-child: Social Networking.

### 1.2.2 Social networks

A technology designed to store a continuously updated copy of the Internet turns out to be pretty good at other things Internet-related. HBase has found a home filling a variety of roles in and around social networking companies. From storing communications between

<sup>12</sup> <http://research.google.com/archive/gfs.html>

<sup>13</sup> <http://research.google.com/archive/mapreduce.html>

<sup>14</sup> <http://research.google.com/archive/bigtable.html>

individuals to communication analytics, HBase has become critical infrastructure at Facebook, Twitter, and StumbleUpon, to name a few.

#### **FACEBOOK**

Hundreds of millions of users send hundreds of billions of messages to each other *every month* using Facebook. Every one of these messages is stored in HBase<sup>15</sup>. Why HBase? In “Hadoop goes realtime at Facebook,”<sup>16</sup> Facebook engineers provide deeper insight into the reasoning behind this decision and their experience using Hadoop and HBase in an online system.

The system supporting Facebook Messages needed to deliver high write throughput, extremely large tables, and strong consistency within a datacenter. In addition to Messages, other applications provided requirements influencing the decision. Read throughput, counter throughput, large tables, and automatic sharding are all necessary features. The engineers found HBase to be an ideal solution because it supports all of these features, has an active user community, and their operations teams have experience with other Hadoop deployments.

#### **TWITTER**

Like many Internet technology companies, Twitter uses a combination of clustered MySQL and memcached. They use HBase to maintain an online backup of their production MySQL databases. HBase enjoys tight integration with Hadoop, allowing the use of MapReduce for analysis over their data without disruption of primary services. HBase is also used for infrastructure monitoring, storing information about the performance of their systems.

#### **Online vs. offline systems**

The terms “online” and “offline” have come up a couple times. For the uninitiated, these terms describe the conditions under which a software system is expected to perform. Online systems have low-latency requirements. Often it is better for these systems to respond with no answer than to take too long producing the correct answer. You can think of a system as online if there’s a user at the other end impatiently tapping their foot. Offline systems do not have this low-latency requirement. There’s a user waiting for an answer, but that response is not expected immediately.

The intent to be an online or an offline system influences many technology decisions when implementing an application. HBase is an online system. Its tight integration with Hadoop MapReduce makes it equally capable of offline processing as well.

#### **STUMBLEUPON**

StumbleUpon came to HBase because they had a counting problem. How do you keep track of the site activity of millions of people? How do you know which site features are most

---

<sup>15</sup> The Underlying Technology of Messages - [https://www.facebook.com/note.php?note\\_id=454991608919](https://www.facebook.com/note.php?note_id=454991608919)

<sup>16</sup> Apache Hadoop goes realtime at Facebook – Borthakur, Sen Sarma, et al. 2011.

popular? How do you use one page view to directly influence the next? StumbleUpon had its start with MySQL but, as the service became more popular, that technology choice failed them. The online demand of this increasing user load was too much for their MySQL clusters and ultimately HBase was chosen to replace those clusters.

At the time, HBase did not directly support the necessary features. StumbleUpon implemented atomic increment in HBase and contributed it back to the project. This contribution introduced HBase into StumbleUpon's production deployment and StumbleUpon into the HBase community. With HBase now a part of their deployment, many new features came to depend upon it and soon HBase became a core component in the StumbleUpon architecture.

Today, HBase powers both online and offline analytics at StumbleUpon. It also is the core technology behind su.pr, StumbleUpon's url shortening service<sup>17</sup>. The company has continued to participate in the HBase community, boasting a number of project committers on their engineering staff. StumbleUpon also built and open sourced OpenTSDB, an infrastructure monitoring application powered by HBase. You'll learn a lot more about HBase when we examine OpenTSDB in the next chapter.

### 1.2.3 Logs

The last use-case for HBase we'll examine is that of log storage. The collection of records describing historical events is common across every industry. Everywhere computing replaces a manual process is a new opportunity to log and learn about that process. Collecting logs is often an online concern while evaluation and analysis can happen offline. HBase plays the critical role of log management at both Mozilla and Trend Micro.

#### MOZILLA

The Mozilla foundation is responsible for the Firefox web browser and Thunderbird email client. These tools are installed on millions of computers worldwide and run on a wide variety of operating systems. When one of these tools crashes, it may send a crash report back to Mozilla in the form of a bug report. How does Mozilla collect these reports? What use are they once collected? The reports are collected via a system called Socorro and are used to direct development efforts toward more stable products. Socorro's data storage and analytics are built on HBase<sup>18</sup>.

The introduction of HBase enabled basic analysis over far more data than was previously possible. This analysis was used to direct developer focus to great effect, resulting in the most bug-free release ever!

#### TREND MICRO

Trend Micro provides Internet security and threat management services to corporate clients. A key aspect of security is awareness, and log collection and analysis is critical for providing

---

<sup>17</sup> HBase at Stumbleupon - [http://www.stumbleupon.com/devblog/hbase\\_at\\_stumbleupon/](http://www.stumbleupon.com/devblog/hbase_at_stumbleupon/)

<sup>18</sup> Moving Socorro to HBase - <http://blog.mozilla.com/webdev/2010/07/26/moving-socorro-to-hbase/>

that awareness in computer systems. Trend Micro uses HBase to manage its web reputation database that requires both row-level updates and support for batch processing with MapReduce. Much like Mozilla's Socorro, HBase is also used to collect and analyze log activity, collecting billions of records every day. The flexible schema in HBase allows their data to easily evolve over time, adding new attributes as analysis processes are refined.

These are just a few examples of how HBase is solving interesting problems new and old. You may have noticed a common thread, using HBase for both online services and offline processing over the same data. This is a role in which HBase is particularly well suited. Now that you have an idea for how HBase can be used, let's get you started using it.

## 1.3 Hello HBase

HBase is built on top of Apache Hadoop and Apache Zookeeper. It's written in Java. To run HBase you'll need to install all of these. Luckily, Cloudera has packaged everything up to turn installation into just a couple commands. In this section you'll get up and running with HBase: install it, create a table, and store some data via the command-line interface. For the purpose of this introduction, we'll assume you're running HBase on Ubuntu Linux Lucid (10.04), the latest long-term support release. You can set up a virtual machine and follow along directly or flip back to Chapter 11 and follow the instructions for your platform and distribution of choice.

### 1.3.1 Quick install

We'll start by installing the Ubuntu Partner and Cloudera repositories. Oracle's Java is the recommended version of the Java runtime for use with HBase. The first command makes Oracle's Java available through apt; the second and third commands provide access to HBase and friends. Open a terminal and follow along:

```
hbase@ubuntu:~$ sudo add-apt-repository \
    "deb http://archive.canonical.com/ lucid partner"
hbase@ubuntu:~$ wget \
    http://archive.cloudera.com/.../cdh3-repository_1.0_all.deb19
hbase@ubuntu:~$ sudo dpkg -i cdh3-repository_1.0_all.deb
```

#### TIP

If you don't already have `apt-add-repository` installed, do so now with the command `sudo apt-get install python-software-properties`.

#### NOTE

Oracle purchased Sun Microsystems in 2009. While Oracle now owns the Java implementation you're using, the package retains the Sun name.

---

<sup>19</sup> The full path is [http://archive.cloudera.com/one-click-install/lucid/cdh3-repository\\_1.0\\_all.deb](http://archive.cloudera.com/one-click-install/lucid/cdh3-repository_1.0_all.deb)

With the repositories registered, update your local cache and install HBase:

```
hbase@ubuntu:~$ sudo apt-get update
hbase@ubuntu:~$ sudo apt-get install sun-java6-jdk hadoop-hbase-master
```

### TIP

Downloading all this software can take a while so be patient!

You can verify everything was installed correctly using the command `apt-cache` command. You should see something along the lines of:

```
hbase@ubuntu:~$ apt-cache policy sun-java6-jdk hadoop-hbase
sun-java6-jdk:
  Installed: 6.26-2lucid1
  Candidate: 6.26-2lucid1
  Version table:
*** 6.26-2lucid1 0
    500 http://archive.canonical.com/ lucid/partner Packages
    100 /var/lib/dpkg/status
hadoop-hbase:
  Installed: 0.90.4+49.1-1~lucid-cdh3
  Candidate: 0.90.4+49.1-1~lucid-cdh3
  Version table:
*** 0.90.4+49.1-1~lucid-cdh3 0
    500 http://archive.cloudera.com/debian/ lucid-cdh3/contrib Packages
    100 /var/lib/dpkg/status
```

The last step is to launch the HBase master. Do so now with the command:  
`sudo /etc/init.d/hadoop-hbase-master start`. Notably, launching the master tells you where it's logging and that it has started ZooKeeper.

```
hbase@ubuntu:~$ sudo /etc/init.d/hadoop-hbase-master start
Starting Hadoop HBase master daemon: starting master, logging to /usr/lib/h
base/logs/hbase-hbase-master-ubuntu.out
12/03/10 19:27:19 INFO server.ZooKeeperServer: Server environment:zookeeper
.version=3.3.3-cdh3u2--1, built on 10/14/2011 04:59 GMT
...
12/03/10 19:27:19 INFO server.ZooKeeperServer: Server environment:os.name=L
inuxhbase-master.
```

That's all there is to it. You just installed HBase in "stand-alone" mode. That means you're running all of Hadoop in one Java process and all of HBase in another. This is how you'll interact with HBase for exploration and local development. You can also run in "pseudo-distributed" mode, a single machine running many Java processes. The last deployment configuration is fully distributed across a cluster of machines. Configuring pseudo- and full-distributed modes is beyond the scope of this intro. Now let's fire up HBase and see what we can see.

### 1.3.2 Interacting with the HBase shell

You use the HBase shell for interacting with HBase from the command line. This works the same way for both local and cluster installations. The HBase shell is a JRuby application wrapping the Java client API. You can run it in either interactive or batch mode. Interactive is for casual inspection of an HBase installation; batch is great for programmatic interaction via shell scripts and or even loading small files. For this section we'll keep to interactive mode.

#### JRuby and JVM languages

Those of you unfamiliar with Java may be confused by this JRuby concept. JRuby is an implementation of the Ruby programming language on top of the Java runtime. In addition to the usual Ruby syntax, JRuby provides support for interacting with Java objects and libraries as well. Java and Ruby aren't the only languages available on the JVM. Jython is an implementation of Python on the JVM and there are entirely unique languages like Clojure and Scala as well. All of these languages can interact with HBase via the Java client API.

Let's start with interactive mode. Launch the shell from your terminal using `hbase shell`. The shell provides you with tab-completion of your commands and inline access to command documentation. Appendix IV replicates that documentation for your convenience.

```
hbase@ubuntu:~$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.90.4-cdh3u2, r, Thu Oct 13 20:32:26 PDT 2011

hbase(main):001:0>
```

If you've made it this far you have confirmed the installation of both Java and the HBase libraries. For the final validation, let's ask for a listing of registered tables. Doing so makes a full-circle request from this client application to the HBase server infrastructure and back again. At the shell prompt type `list` and press `enter`. You should see zero results found as a confirmation and you'll be again greeted with a prompt.

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.5710 seconds

hbase(main):002:0>
```

With your installation complete and verified, let's create a table and store some data.

### 1.3.3 Storing Data

HBase uses the *table* as the top-level structure for storing data. To write data into HBase you need a table to write it into. To begin, create a table called `mytable` with a single *column family*. Yes, "column family." Don't worry, we'll explain later. Create your table now.

```

hbase(main):002:0> create 'mytable', 'c'      #A
0 row(s) in 1.0730 seconds
hbase(main):003:0> list                      #B
TABLE                                         #B
mytable                                       #B
1 row(s) in 0.0080 seconds

```

### WRITING DATA

With a table created you can now write some data. Let's add the string "hello HBase" to our table. In HBase parlance we say "put the bytes 'hello HBase' to a cell in 'mytable' in the 'first' row at the 'c:message' column." Catch all that? Again, we'll cover all this terminology in a later chapter. For now, perform the write.

```

hbase(main):004:0> put 'mytable', 'first', 'c:message', 'hello HBase'
0 row(s) in 0.2070 seconds

```

That was easy. HBase stores numbers just as well as strings. Go ahead and add a couple more values, like so:

```

hbase(main):005:0> put 'mytable', 'second', 'c:hex', 0x0
0 row(s) in 0.0130 seconds
hbase(main):006:0> put 'mytable', 'third', 'c:double', 3.14159
0 row(s) in 0.0080 second

```

You now have three cells in three rows in your table. Notice you didn't define the columns before you used them. Nor did you specify what type of data you stored in each column. This is what the NoSQL crowd means when they say HBase is a "schemaless" database. But what good is writing data if you cannot read it? No good at all. It's time to read your data back.

### READING DATA

HBase gives you two ways to read data: get and scan. As you astutely noticed, the command you gave HBase to store those cells was put. get is the complement of put, reading back a single cell from a single column in a single row. Remember before when we mentioned HBase having a key-value API but with a twist? scan is that twist. Chapter 3 will explain how scan works and why it's important. In the mean time, just focus on using it. We'll start with get.

```

hbase(main):007:0> get 'mytable', 'first'
COLUMN      CELL
c:message   timestamp=1323483954406, value=hello HBase
1 row(s) in 0.0250 seconds

```

Just like that, you pulled out your first row. The shell shows you all the cells in the row, organized by column, with the value associated at each *timestamp*. HBase supports versioning of data. Enabled by default, you can write new values into the same cell and HBase will keep track of each of them. It will only return the latest value for the requested unless you specify otherwise.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=802>

Use `scan` when you want values from multiple rows. Be careful! Unless you specify otherwise it will return all rows in the table. Don't say we didn't warn you. Try it out!

```
hbase(main):008:0> scan 'mytable'
ROW          COLUMN+CELL
  first      column=c:message, timestamp=1323483954406, value=hello HBase
  second     column=c:hex,      timestamp=1323483964825, value=0
  third      column=c:double,   timestamp=1323483997138, value=3.14159
3 row(s) in 0.0240 seconds
```

All your data came back. Excellent. Notice the order in which HBase return your rows. They're ordered by the row name; HBase calls this the *rowkey*. HBase has a couple other tricks up its sleeve but everything else is built on the basic concepts you've just used. Easy? Easy. Let's wrap it up.

## 1.4 Summary

We covered quite a bit of material for an introductory chapter. Knowing where a technology comes from is always helpful when you're learning. By now you should understand the roots of HBase and have some context for this whole NoSQL phenomenon. You also understand the basics of the problem HBase is designed for as well as some of the problems HBase has solved. Not only that but you're running HBase *right now* and using it to store some of your *most precious* "hello world" data.

No doubt we've raised more questions for you. *Why is strong consistency important? How does the client find the right node for a read? What's so fancy about scans anyway? What other tricks are up its sleeve?* We're so glad you asked! We'll answer all these and more in the coming chapters. The next chapter will break down an application built on HBase, starting with its high-level design, examining the HBase schema, diving into the code. You will gain a sense for how to use HBase in your own applications.