Chapter 17: Administering Your Hadoop Cluster

In This Chapter

- Seeing why having a well-running Hadoop cluster is good for you
- Exploring administration commands
- Improving performance and setting benchmarks
- Planning for when things go wrong
- Working with Apache Hadoop's Capacity Scheduler
- Dealing with security issues
- Adding resources to your administrator toolset

You'll want to keep your Hadoop cluster running smoothly and at a high level of performance. For that to happen, you need to master the mysteries of Hadoop administration. Part of this process involves careful planning to ensure that you deploy and configure appropriate hardware for your Hadoop cluster, the use of judicious benchmarking to evaluate performance, and a good understanding of the anticipated workloads.

Complicating matters a bit is the fact that not only is most of the Hadoop ecosystem quite compartmentalized, but each component also has its own administrative issues. We deal with these issues in various sections throughout this book, where appropriate. This chapter (Chapter 17) introduces you to more general administrative concepts.

Achieving Balance: A Big Factor in Cluster Health

A cluster is said to be *balanced* if no under- or overutilized slave nodes are in the cluster. In this context, a *utilization level* is defined in terms of the percentage of space that's used. A Hadoop cluster can become imbalanced whenever a major change occurs — say, when a slave node is added to the cluster. An imbalanced cluster can lead to bandwidth problems and reduced *read parallelism* (where many applications can read data independently, instead of having to wait their turn), and a Hadoop administrator should be prepared to redistribute data blocks when cluster imbalance occurs.

The goal, then, is to spread data as uniformly as possible across the slave nodes in the cluster. As much as this idea seems to make obvious sense, it isn't always achievable. When a slave node is added to an existing cluster, the NameNode must choose which existing slave nodes are to receive some of the new data blocks. One goal is to place different replicas of a particular block across server racks to minimize the loss of an entire rack. Another goal is to reduce network I/O by placing one replica on the same rack as the node that's writing to a file.

TIP Despite your best-laid plans, various competing factors might cause new data to be placed across the slave nodes in a non-uniform manner. Luckily, one tool can analyze block placement and rebalance data across the slave nodes for you: the Hadoop `balancer` command, which gets a nice mention in the following section, in Table 17-1.

Mastering the Hadoop Administration Commands

Any Hadoop administrator worth his salt must master a comprehensive set of commands for cluster administration. Table 17-1 summarizes the most important commands. Know them, and you will advance a long way along the path to Hadoop wisdom. Table 17-2 summarizes the Hadoop `dfsadmin` command options.

Table 17-1: Administration Commands

Command	*What It Does*	*Syntax*	*Exampl*
`balancer`	Runs the cluster-balancing utility. The specified threshold value, which represents a percentage of disk capacity, is used to overwrite the default threshold value (10 percent). To stop the rebalancing process, press Ctrl+C.	`hadoop balancer [-threshold <threshold>]`	`hadoo`
`daemonlog`	Gets or sets the log level for each daemon. Connects to http://host:port/logLevel?log=name `and prints or sets the log`	hadoop daemonlog -getlevel <host:port> <name>; hadoop daemonlog	hadoop d
	level of the daemon that's running at *`host:port`*. Hadoop daemons generate log files that help you determine what's happening on the system, and you can use the `daemonlog` command to temporarily change the log level of a Hadoop component when you're debugging the system. The change becomes effective when the daemon restarts.	`-setlevel <host:port><name> <level>`	`org.a` `-setl` `10.25` `DEBUG`
`datanode`	Runs the HDFS DataNode service, which coordinates storage on each slave node. If you specify `-rollback`,	`hadoop datanode [-rollback]`	`hadoo`

	the DataNode is rolled back to the previous version. Stop the DataNode and distribute the previous Hadoop version before using this option.		
`dfsadmin`	Runs a number of Hadoop Distributed File System (HDFS) administrative operations. Use the -help option to see a list of all supported options. The generic options are a common set of options supported by several commands. (For detailed information about generic options, visit http://hadoop.apache.org/docs/r2.0.5-alpha/hadoop-project-dist/hadoop-common/CommandsManual.html. For detailed information about the individual `dfsadmin` command options, see Table 17-2.)	`hadoop dfsadmin [GENERIC_OPTIONS] [-report] [-safemodeenter | leave | get | wait] [-refreshNodes] [-finalizeUpgrade] [-upgradeProgress status | details | force] [-metasave filename] [-setQuota <quota> <dirname>…<dirname>] [-clrQuota <dirname>…<dirname>] [-restoreFailed Storage true|false|check] [-help [cmd]]`	
`mradmin`	Runs a number of MapReduce administrative operations. Use the `-help` option to see a list of all supported options. Again, the generic options are a common set of options that are supported by several commands. (For detailed information about these options, visit http://hadoop.apache.org/docs/r2.0.5-alpha/hadoop-project-dist/hadoop-common/CommandsManual.html.) If you specify `-refreshServiceAcl`, `mradmin` reloads the service-level authorization policy file (JobTracker reloads the authorization policy file); `-refreshQueues` reloads the queue access control lists (ACLs) and state (JobTracker reloads the `mapred-queues.xml` file); `-refreshNodes` refreshes the hosts information at the JobTracker; `-refreshUserToGroupsMappings` refreshes user-to-groups mappings; `-refreshSuperUserGroupsConfiguration` refreshes superuser proxy groups mappings; and `-help [cmd]` displays help for the given command or for all commands if none is specified.	`hadoop mradmin [ GENERIC_OPTIONS ][-refresh ServiceAcl] [-refreshQueues] [-refreshNodes] [-refreshUserToGroupsMappings][-refreshSuperUserGroupsConfiguration] [-help [cmd]]`	`hadoo`
`jobtracker`	Runs the MapReduce JobTracker node, which coordinates the data processing system for Hadoop. If you specify `-dumpConfiguration`, the configuration that's used by the JobTracker and the queue configuration in JSON format are written to standard output.	`hadoop jobtracker [-dumpConfiguration]`	`hadoo`
`namenode`	Runs the NameNode, which coordinates the storage for the whole Hadoop cluster. If you specify `-format`, the NameNode is started, formatted, and then stopped; with `-upgrade`, the NameNode starts with the upgrade option after a new Hadoop version is distributed; with `-rollback`, the NameNode is rolled back to the previous version (remember to stop the cluster and distribute the previous Hadoop version before using this option); with `-finalize`, the previous state of the file system is removed, the most recent upgrade becomes permanent, rollback is no longer available, and the NameNode is stopped; finally, with `-importCheckpoint`, an image is loaded from the checkpoint directory (as specified by the `fs.checkpoint.dir` property) and saved into the current directory.	`hadoop namenode [-format] | [-upgrade] | [-rollback] | [-finalize] | [-importCheckpoint]`	`hadoo`
`secondarynamenode`	Runs the secondary NameNode. If you specify `-checkpoint`, a checkpoint on the secondary NameNode is performed if the size of the EditLog (a transaction log that records every change that occurs to the file system metadata) is greater than or equal to `fs.checkpoint.size`; if you specify `force`, a checkpoint is performed regardless of the EditLog size; specify `geteditsize` and the EditLog size is printed.	`hadoop secondarynamenode [-checkpoint [force]] | [-geteditsize]`	`hadoo`
`tasktracker`	Runs a MapReduce TaskTracker node.	`hadooptasktracker`	`hadoo`

Table 17-2: The Hadoop `dfsadmin` Command

Option	*What It Does*
`-report`	Reports basic file system information and statistics.

`-safemode enter |__leave | get | wait`	Manages *safe* mode, a NameNode state in which changes to the name space are not accepted and blocks can be neither replicated nor deleted.
	The NameNode is in safe mode during start-up so that it doesn't prematurely start replicating blocks even though there are already enough replicas in the cluster.
`-refreshNodes`	Forces the NameNode to reread its configuration, including the `dfs.hosts.exclude` file. The NameNode decommissions nodes after their blocks have been replicated onto machines that will remain active.
`-finalizeUpgrade`	Completes the HDFS upgrade process. DataNodes and the NameNode delete working directories from the previous version in order to keep things nice and neat.
`-upgradeProgress status | details | force`	Requests the standard or detailed current status of the distributed upgrade, or forces the upgrade to proceed.
`-metasave filename`	Saves the NameNode's primary data structures to *`filename`* in a directory that's specified by the `hadoop.log.dir` property. File *`filename`*, which is overwritten if it already exists, contains one line for each of these items: a) DataNodes that are exchanging heartbeats (electronic "signs of life") with the NameNode; b) blocks that are waiting to be replicated; c) blocks that are being replicated; and d) blocks that are waiting to be deleted.
`-setQuota <quota> <dirname>…<dirname>`	Sets an upper limit on the number of names in the directory tree. You can set this limit (a long integer) for one or more directories simultaneously.
`-clrQuota <dirname>… <dirname>`	Clears the upper limit on the number of names in the directory tree. You can clear this limit for one or more directories simultaneously.
`-restoreFailedStorage true | false | check`	Turns on or off the automatic attempts to restore failed storage replicas. If a failed storage location becomes available again, the system attempts to restore edits and the `fsimage` during a checkpoint. The `check` option returns the current setting.
`-help [cmd]`	Displays help information for the given command or for all commands if none is specified.

Understanding Factors for Performance

Many factors affect the performance of a Hadoop cluster, including the hardware configuration of machines in the cluster, the software configuration, and how well the map and reduce tasks are tuned for the particular jobs they perform when processing your workloads. This section takes a look at each one of these factors in turn.

Hardware

As you might expect, because each node in a Hadoop cluster is used to store (DataNode) *and* process (TaskTracker) data, the hardware should be configured with both roles in mind. Always use the fastest machines you can afford, with processing speed a function of the number of cores available. Also, remember that having lots of RAM minimizes the number of times that data must be read from disk. RAM requirements for the NameNode increase in proportion to the total number of data blocks in the cluster, and extra RAM on the NameNode accommodates the future growth of the cluster. Disk speed affects the degree of throughput that can be achieved, and the number of disks per node affects the cluster's ability to "scale up," which in this case means the ability to add storage to individual nodes in the system.

MapReduce

TIP Tuning the number of map tasks and reduce tasks for a particular job in your workload is another way that you can optimize performance, because each task has a significant level of overhead that can represent a significant cost for you when the length of time spent on task execution ends up being relatively short.

If your jobs involve larger data sets, increasing the block size reduces the number of tasks, which also has a positive impact on performance.

When planning to maximize the performance of your Hadoop cluster, you often have to make a trade-off between the overhead of data movement and your IO (input/output) costs. If your nodes have local storage disks, it might make sense to move MapReduce processing to those nodes so that input/output is minimized. If, on the other hand, the data isn't available locally, you have to move it to the nodes where processing will occur. This situation can result in network congestion and eroded performance when data volumes are very large. Although data replication can address this issue by producing a local copy of the data at each processing node, creating, distributing, and storing replicas in a large cluster can be quite costly.

Benchmarking

After you have defined the types of workloads to run on your system, you can begin to benchmark those workloads to identify input/output and data processing bottlenecks.

So, what exactly is benchmarking? Many types of benchmarking are out there, but when the term is applied to a Hadoop cluster, what's usually meant is a process whereby you compare the cluster's performance either to previously measured values or to published best-of-breed values. Performance benchmarking involves the monitoring of specific indicators (for example, throughput, response time) under controlled conditions. Benchmarking is typically an ongoing process that's designed to promote continuous improvement.

When you have set up a new cluster, benchmarking is a good way to determine whether the cluster was set up correctly. See whether you get the expected results. Your expectations might be driven by published results from other clusters that were configured in a similar way. You can also tune the cluster by comparing monitored results with benchmark values.

TIP To produce the best results, run benchmarks when your cluster isn't being used by others.

It just so happens that the Hadoop distribution includes a number of benchmarks you can use. Examples include TestDFSIO, NNBench, and MRBench (in `hadoop-*test*.jar`) and TeraSort (in `hadoop-*examples*.jar`). If you're curious about what these benchmarks can offer, check out this list:

- **TestDFSIO:** The TestDFSIO benchmark is useful for testing the I/O performance of the HDFS. This benchmark uses a MapReduce job to read and write files in separate map tasks, whose output is used for collecting statistics that are accumulated in the reduce tasks to produce a summary result. The benchmark data is then appended to a local file named `TestDFSIO_results.log` and written to standard output.
- **NNBench:** The NNBench benchmark is useful for load-testing the NameNode. This benchmark simulates a high volume of file manipulation requests against the HDFS to "stress-test" the NameNode's ability to manage the HDFS.
- **MRBench:** The MRBench benchmark loops small jobs to determine whether they're running efficiently. It's used to test the MapReduce layer.
- **TeraSort:** The TeraSort benchmark sorts a fixed amount of data as quickly as possible. This benchmark tests both the HDFS and MapReduce layers of your Hadoop cluster and is useful for comparing the performance of your cluster with other clusters. You can use the TeraSort benchmark to fine-tune your Hadoop configuration after running the TestDFSIO benchmark.

To get a list of the benchmarks that come with Hadoop, run the JAR file with no arguments:

```
% hadoop jar $HADOOP_INSTALL/hadoop-*-test.jar
```

To retrieve usage information for a specific benchmark, run the benchmark with no arguments. For example:

```
% hadoop jar $HADOOP_INSTALL/hadoop-*-test.jar TestDFSIO
```

TIP When tuning the cluster, be sure to include jobs that are similar to the workloads you'll run most often. The standard benchmarks that come with Hadoop are fine in general, but tune the cluster for your specific workloads. And remember to test the same set of jobs and data every time so that you can meaningfully compare runs.

Tolerating Faults and Data Reliability

The glory of Apache Hadoop is that, in a Hadoop cluster, data is distributed across a number of balanced machines, and replication is used to ensure both data reliability and fault tolerance.

By default, each block is replicated to three slave nodes. The *replication factor* is configurable. Block replication is maintained by the system automatically. The NameNode is responsible for detecting failed slave nodes or unavailable replicas and ensures that usable replicas are copied to other nodes.

REMEMBER The DataNode service on each slave node sends heartbeats (indicating their good health) to the NameNode by using the same port number that was defined for the NameNode daemon (typically, TCP 9000 or TCP 8020). A *heartbeat* is a periodic signal in the form of a TCP handshake, which is the procedure that takes place between two TCP/IP nodes to establish a connection. As you might expect, regular heartbeats from a DataNode tell the NameNode that the DataNode is alive and well. By default, the heartbeat interval is three seconds, and if the NameNode doesn't receive a heartbeat from a particular DataNode within ten minutes, the DataNode is presumed to be "dead," and its blocks are scheduled for replication on other nodes.

TIP Keep the heartbeat frequency high, even on big clusters. NameNodes can handle thousands of heartbeats per second without difficulty, and the granularity of the information that is provided in this way is essential to maintaining good cluster health.

Every tenth heartbeat from a particular DataNode is a *block report,* by which the DataNode identifies its blocks to the NameNode. This information is used by the NameNode to determine whether the correct number of block replicas exists. If a DataNode is dead, its data is of course unavailable, but the NameNode is aware of which replicas died along with the node and can replicate those blocks to other slave nodes.

You expect your Hadoop cluster to be always available. One way to make it happen is to configure the HDFS High Availability (HA) feature, using a shared NFS directory.

Prior to the Hadoop 2.x series, the NameNode was a *single point of failure* in an HDFS cluster — in other words, if the machine on which the single NameNode was configured became unavailable, the entire cluster would be unavailable until the NameNode could be restarted. This was bad news, especially in the case of unplanned outages, which could result in significant downtime if the cluster administrator weren't available to restart the NameNode.

The solution to this problem is addressed by the HDFS High Availability feature. The idea is to run two NameNodes in the same cluster — one active NameNode and one hot standby NameNode. If the active NameNode crashes or needs to be stopped for planned maintenance, it can be quickly *failed over* to the hot standby NameNode, which now becomes the active NameNode. The key is to keep the standby node synchronized with the active node; this action is now accomplished by having both nodes access a shared NFS directory. All namespace changes on the active node are logged in the shared directory. The standby node picks up those changes from the directory and applies them to its own namespace. In this way, the standby NameNode acts as a current backup of the active NameNode. The standby node also has current block location information, because DataNode heartbeats are routinely sent to both active and standby NameNodes.

To ensure that only one NameNode is the "active" node at any given time, configure a *fencing process* for the shared storage directory; then, during a failover, if it appears that the failed NameNode still carries the active state, the configured fencing process prevents that node from accessing the shared directory and permits the newly active node (the former standby node) to complete the failover.

REMEMBER The machines that will serve as the active and standby NameNodes in your High Availability cluster should have equivalent hardware. The shared NFS storage directory, which must be accessible to both active and standby NameNodes, is usually located on a separate machine and can be mounted on each NameNode machine. To prevent this directory from becoming a single point of failure, configure multiple network paths to the storage directory, and ensure that there's redundancy in the storage itself. Use a dedicated network-attached storage (NAS) appliance to contain the shared storage directory.

Putting Apache Hadoop's Capacity Scheduler to Good Use

Although it might seem that Hadoop is an inherently limitless resource, there are limits to its capacity, and cluster resources must be managed appropriately to avoid performance issues. You don't have to be an organization such as Yahoo! or Facebook, which control some of the largest Hadoop clusters in the world, to appreciate the need for capacity management.

Apache Hadoop's Capacity Scheduler was designed to address — you guessed it — capacity management. The Capacity Scheduler, a pluggable scheduler and console for Hadoop, uses *job queues* to facilitate the organized sharing of Hadoop clusters. It guarantees minimum capacity levels for all queues and makes unused capacity available to overloaded queues, which leads to optimized cluster utilization.

The Capacity Scheduler provides a set of limits to ensure that a single application cannot consume a disproportionate amount of cluster resources, thereby promoting fairness and stability.

You can assign jobs to specific queues and, as an administrator, define each queue's *maximum capacity* — a limit on the amount of resources a queue can claim beyond its guaranteed capacity.

Each queue enforces additional restrictions, including a limit on

- The resources that a specific user can access if multiple users are accessing the queue at the same time
- The number of accepted or active jobs per queue or per user
- The number of pending tasks per queue or per user

Moreover, *hierarchical queues* ensure that resources are shared among an organization's subqueues before another organization's queues are allowed to access unused resources.

From a security perspective, each queue has access control lists (ACLs) that control which users are authorized to submit applications to specific queues. Moreover, users cannot view or change the applications of other users.

As an administrator, you can change queue definitions, properties, and ACLs at run time, but you cannot delete queues. You can, however, stop a queue at run time to block the submission of new applications while existing applications are running. Because existing applications continue to run, the queue is able to run its course. Administrators can also start any stopped queues.

TECHNICAL STUFF To take advantage of the Capacity Scheduler, you have to configure your ResourceManager (see Chapter 7) to use it. To do so, set the `yarn.resourcemanager.scheduler.class` property in the `conf/yarn-site.xml` file to

```
org.apache.hadoop.yarn.server.resourcemanager.scheduler.
           capacity.CapacityScheduler
```

The configuration file for the Capacity Scheduler is `conf/capacity-scheduler.xml`. You can edit this file to define new queues or to modify existing ones. After editing the configuration file, run `yarn rmadminrefreshQueues`, as shown here:

```
$ vi $HADOOP_CONF_DIR/capacity-scheduler.xml
$ $HADOOP_YARN_HOME/bin/yarn rmadmin -refreshQueues
```

The Capacity Scheduler has the predefined queue named `root`. All other queues are defined as children of the `root` queue.

It's easy to define new queues: Simply configure the `yarn.scheduler.capacity.root.queues` property with a list of child queue names, separated by commas. For example, to add two child queues (`q1` and `q2`), you'd do this:

```
<property>
 <name>yarn.scheduler.capacity.root.queues</name>
 <value>q1,q2</value>
 <description>The child queues under root.
 </description>
</property>
```

The queue hierarchy is denoted by a path notation (starting with `root`) in which each queue is separated by a dot: `yarn.scheduler.capacity.queue-path.queues`. For example:

```
<property>
```

```
 <name>yarn.scheduler.capacity.root.q1.queues</name>
 <value>q1a1,q1a2</value>
 <description>The child queues under q1.
 </description>
</property>
```

TIP For more information about the Capacity Scheduler, visit

```
http://hadoop.apache.org/docs/current/hadoop-
          yarn/hadoop-yarn-site/CapacityScheduler.html
```

Setting Security: The Kerberos Protocol

When we speak about security in a Hadoop context, we are referring to an *authentication* method to ensure that users of the Hadoop cluster are who they say they are. File system permissions, which enforce *authorization,* are designed to control the file operations (such as read or write) that a specific user or group can perform. For various reasons, including the importance of protecting sensitive data from users who don't have a business need to access such data, shared clusters, including Hadoop clusters, must have effective authentication mechanisms in place.

Secure authentication of Hadoop clusters has been available through the Kerberos protocol since Hadoop 2. Kerberos is a mature, open source computer network authentication protocol that enables nodes to securely verify their identity to one another. Kerberos does not manage file or directory permissions.

The Kerberos protocol is implemented as a series of negotiations between a client, the authentication server (AS), and the service server (SS). This is what happens, in a nutshell: When a user logs on, the client authenticates itself to the AS, which sends the username to a key distribution center (KDC). The KDC then issues a time-stamped ticket-granting ticket (TGT), which is encrypted and returned to the client.

When the client wants to communicate with another node, it sends the TGT to a ticket-granting server (TGS), which verifies that the TGT is valid. The TGS then issues a ticket and session keys, which are returned to the client. The client, in turn, sends the ticket and a service request to the service server (SS), which, in the case of a Hadoop cluster, might be the NameNode or the JobTracker.

A TGT expires after a certain period (ten hours, by default) but can be renewed for as long as a week. You can provide a single sign-on to Hadoop by automating the authentication process at operating system login.

To use Kerberos authentication with Hadoop, you must install and configure a key distribution center. Enable Kerberos authentication by setting the `hadoop.security.authentication` property in `core-site.xml` to `kerberos`. Enable service-level authorization by setting the `hadoop.security.authorization` property in the same file to `true`. You should also configure access control lists (ACLs) in the `hadoop-policy.xml` configuration file to specify which users and groups have permission to connect to the various Hadoop services, such as NameNode communication.

Expanding Your Toolset Options

You are not alone. Lots of very smart people have come up with a bunch of tools and interfaces that you can use to make administering a Hadoop cluster easier. Two of the more prominent tools are Hue and Ambari, which we highlight in the upcoming sections.

Hue

Hue is a browser-based graphical user interface to Apache Hadoop. Folks just call it Hue these days, but the name originates as an acronym for Hadoop User Experience.

Hue was initially developed as an open source project by Cloudera. Although Hue comes bundled with Cloudera (and with many Hadoop distributions to boot), it's also available from GitHub as open source code. With Hue, you can browse the HDFS (by using FileBrowser); create and manage user accounts; monitor your cluster; submit and view MapReduce or YARN jobs (by using JobSub and JobBrowser); enable a user interface (named Beeswax) to Hive; use an HBase browser; access query editors for Hive, Pig, Cloudera Impala, and Sqoop 2; and much more.

Table 17-3 summarizes the various components that make up the Hue offering.

Table 17-3: The Components of Hue, a Graphical User Interface to Apache Hadoop

Component	*What You Can Do with It*
File Browser	Upload, browse, and manipulate files and directories in the Hadoop Distributed File System (HDFS).
HBase Browser	Quickly access very large tables, create new tables, add data, or modify existing data.
Cloudera Search	Search for data that's stored in the HDFS or HBase. SQL and programming skills aren't required, because Cloudera Search provides a simple, full-text interface for searching.
Job Designer	Create and submit MapReduce, YARN, or Java jobs to your Hadoop cluster.
Job Browser	Monitor the MapReduce or YARN jobs that are running on your Hadoop cluster. Jobs appear in a list, and you can link to a list of tasks for a specific job. You can view task details or logs to troubleshoot failed jobs.
Metastore Manager	Manage the databases, tables, and partitions of the Hive metastore, which is shared by Beeswax and Cloudera Impala. You can use the

	Metastore Manager to create or drop a database; create, browse, or drop tables; or import data into tables. (For more on Apache Hive, see Chapter 13.)
Beeswax Hive User Interface	Run and manage queries on Apache Hive, a distributed data warehouse for data that's stored in the HDFS. You can download query results in a Microsoft Office Excel worksheet or a text file.
Sqoop 2	Efficiently move large amounts of data between relational databases and the HDFS.
Cloudera Impala	Issue low-latency SQL queries against data that's stored in the HDFS or HBase without the need for data movement or transformation. This massively parallel processing query engine runs natively on Apache Hadoop.
Pig Editor	Edit your Pig scripts with autocompletion and syntax highlighting. (For more on Pig and Pig scripting, see Chapter 8.)
Oozie Editor and Dashboard	Define Oozie workflow and coordinator applications, run workflow and coordinator jobs, and view the status of those jobs. (For more on Oozie, see Chapter 10.)
Zookeeper User Interface	Browse the Znode hierarchy of your Zookeeper cluster, and add, edit, or delete Znodes. (For more on Zookeeper, see Chapter 12.)
User Admin	Add, delete, and manage Hue users or groups (if you're the administrator); add users or groups individually or import them from an LDAP directory. Granted permissions determine which Hue applications, or application features, users or groups can access.

Hue also comes with a software development kit (SDK) that enables you to reuse Hue libraries and build applications on top of Hadoop.

Hue is designed to enhance the Hadoop user experience by facilitating real-time interaction with data and helping you to get results faster. It's intended to be used by a variety of users and is offered in several languages, including Spanish, French, German, Portuguese, Brazilian Portuguese, Japanese, simplified Chinese, and Korean.

For information about getting started with Hue, including development prerequisites, visit one of these sites:

```
https://github.com/cloudera/hue
http://cloudera.github.io/hue/docs-2.0.1/manual.html
```

Ambari

Apache Ambari is a tool for provisioning, configuring, managing, and monitoring Apache Hadoop clusters. With Ambari, you can deploy and operate a complete Hadoop stack by using a browser-based management interface.

Apache Ambari is still undergoing *incubation* at the Apache Software Foundation (ASF); incubation is required of all newly accepted projects until their infrastructure is deemed consistent with other successful ASF projects.

The Apache Ambari project is designed to simplify Hadoop management by providing a set of simple GUI tools. It now supports the following Hadoop components: the HDFS, MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig, and Sqoop.

Ambari makes it easy for system administrators to perform the tasks described in this list:

- Provision a Hadoop cluster:
  - *Ambari provides an easy-to-use wizard to help you install Hadoop services*.
  - *Ambari handles the configuration of Hadoop services for your cluster*.
- Manage a Hadoop cluster:
  - *Ambari provides central management for starting, stopping, and reconfiguring Hadoop services across your entire cluster*.
- Monitor a Hadoop cluster:
  - *Ambari provides a dashboard for monitoring the health and status of your Hadoop cluster*.
  - *Ambari leverages Ganglia for metrics collection. Ganglia, a BSD-licensed open source project, is a scalable distributed monitoring system for high-performance computing systems such as clusters. For information about Ganglia monitoring, visit* http://ganglia.sourceforge.net.
  - *Ambari leverages Nagios for system alerting and sends e-mails when your attention is needed, such as when a node fails. Nagios, an open source application, offers monitoring and alerting services for servers, switches, applications, and services. For information about Nagios monitoring, visit* www.nagios.org.

Ambari also helps application developers and system integrators integrate Hadoop provisioning, management, and monitoring capabilities into their own applications by using Ambari's Representational State Transfer (REST) APIs. (REST is an architectural style for client/server communication over HTTP.)

Ambari now supports the 64-bit version of these operating systems:

- RHEL (Redhat Enterprise Linux) 5 and 6
- CentOS 5 and 6

- OEL (Oracle Enterprise Linux) 5 and 6
- SLES (SuSE Linux Enterprise Server) 11

For more information about the Apache Ambari, visit one of these sites:

```
http://incubator.apache.org/ambari
http://hortonworks.com/hadoop/ambari
```

Hadoop User Experience (Hue)

Hadoop User Experience (Hue, for short) is a browser-based graphical user interface to Apache Hadoop. You can use Hue to

- Browse the HDFS
- Create and manage user accounts
- Monitor the cluster
- Submit and view MapReduce or YARN jobs (by using JobSub and JobBrowser)
- Enable Beeswax, an aptly named user interface for Apache Hive, which is Hadoop's data warehouse infrastructure with SQL-like features
- Use an HBase browser
- Access query editors for (the aforementioned) Hive, Pig, Cloudera Impala (a query engine with SQL capabilities) and Sqoop 2

Hue was developed as an open source project by — and is available from — Cloudera (www.cloudera.com/). The current version of Hue is 2.3.0.

Table 17-4 summarizes the various components that come packaged with Hue.

Table 17-4: Hue Components

Component	*What You Can Do with It*
File Browser	Upload, browse, and manipulate files and directories in the Hadoop distributed file system (HDFS).
HBase Browser	Quickly access very large tables, create new tables, add data, or modify existing data.
Cloudera Search	Search for data that's stored in the HDFS or HBase. SQL and programming skills aren't required, because Cloudera Search provides a simple, full-text interface for searching.
Job Designer	Create and submit MapReduce, YARN, or Java jobs to your Hadoop cluster.
Job Browser	Monitor the MapReduce or YARN jobs that are running on your Hadoop cluster. Jobs appear in a list, and you can link to a list of tasks for a specific job. You can view task details or logs to troubleshoot failed jobs.
Metastore Manager	Manage the databases, tables, and partitions of the Hive metastore that are shared by Beeswax and Cloudera Impala. You can use the Metastore Manager to create or drop a database; create, browse, or drop tables; and import data into tables.
Beeswax Hive User Interface	Run and manage queries on Apache Hive. You can download query results in a worksheet or text file in Microsoft Office Excel. (For more on Hive, see Chapter 13.)
Sqoop 2	Efficiently move large amounts of data between relational databases and the HDFS.
Cloudera Impala	Issue low-latency SQL queries against data stored in the HDFS or in HBase without the need for data movement or transformation. (This massively parallel processing query engine runs natively on Apache Hadoop.)
Pig Editor	Edit Pig scripts with autocompletion and syntax highlighting. (For much more on Pig, see Chapter 8.)
Oozie Editor and Dashboard	Define Oozie workflow and coordinator applications, run workflow and coordinator jobs, and view the status of those jobs. (For more on Oozie, check out Chapter 10.)
Zookeeper User Interface	Browse the Znode hierarchy of the Zookeeper cluster, and add, edit, or delete Znodes. (You can find lots more on Zookeeper in Chapter 13.)
User Admin	Add, delete, and manage Hue users or groups (if you're the administrator). You can add users or groups individually or import them from an LDAP directory, such as a corporate e-mail directory. Granted permissions determine which Hue applications or application features can be accessed by users or groups.

Hue also comes supplied with an SDK (System Development Kit) that enables you to reuse Hue libraries and build applications on top of Hadoop.

Hue is designed to enhance the Hadoop user experience by facilitating real-time interaction with data and by helping you get results faster. Intended to be used by a variety of users, Hue is offered in several languages, including Spanish, French, German, Portuguese, Brazilian Portuguese, Japanese, simplified Chinese, and Korean.

TIP For information about getting started with Hue, including development prerequisites, visit one of these sites:

```
https://github.com/cloudera/hue
http://cloudera.github.io/hue/docs-2.0.1/manual.html
```

The Hadoop Shell

The *Hadoop shell* is a family of commands that you can run from your operating system's command line. The shell has two sets of commands: one for file manipulation (similar in purpose and syntax to Linux commands that many of us know and love) and one for Hadoop administration. For a detailed description of the file management commands available in the Hadoop shell, see the section in Chapter 5 about managing files with the Hadoop file system commands. For details on the administration commands in the Hadoop shell, see the "Mastering the Hadoop Administration Commands" section, earlier in this chapter.

Basic Hadoop Configuration Details

In Hadoop 0.19.x or earlier, you had to modify only one configuration file, `hadoop-site.xml`, in order to lay the groundwork for your Hadoop project. In Hadoop 0.21 and later, however, you have a bit more work coming your way. More specifically, you need to configure three separate XML files, all found in the `HADOOP_HOME/conf` directory:

- `core-site.xml`
- `hdfs-site.xml`
- `mapred-site.xml`

Your Hadoop configuration is driven by two distinct types of configuration files:

- **Default (read-only):** Files here include `src/core/core-default.xml, src/hdfs/hdfs-default.xml`, and `src/mapred/mapred-default.xml`.
- **Site-specific configuration:** Files here include `conf/core-site.xml, conf/hdfs-site.xml`, and `conf/mapred-site.xml`.

REMEMBER These files are also known as resources. A *resource* contains a set of name/value pairs as XML data. Each resource is identified by either a string value or a path. If you specify a string value, the classpath is searched for a file whose name matches that value. If you specify a path, the local file system is searched directly. The default resources (XML files), which are read-only, reside inside the `hadoop-common` and `hadoop-hdfs` JAR files. These files, which are read from the JAR files directly, should never be modified.

The site-specific resources are loaded from the classpath, and their values are used to override the corresponding values in the matching default resource. If you've surmised that the previous statement implies that the default resource is loaded first, followed by the site-specific resource, you're right! For example:

- `core-default.xml`: Contains read-only default values for your Hadoop configuration and is read in first.
- `core-site.xml`: Contains site-specific configuration values for your Hadoop deployment and is read in second; contains only those values that need to be changed from the default.

The following code example shows a configuration specification from a `core-site.xml file`:

```
<property>
 <name>hadoop.tmp.dir</name>
 <value>/home/hadoop/hadoop-0.20.2/hdfs-tmp</value>
 <description>A base for other temporary
              directories.</description>
</property>
```

Note that each `<property>` element (`<name>`, `<value>,` and `<description>` in this example) defines a specific configuration name/value pair. The file can contain any number of these `<property>` elements, which are enclosed by one `<configuration>` element, as in this example:

```
<configuration>
...
<property>
 <name>...</name>
 <value>...</value>
 <description>...</description>
</property
...
</configuration>
```

The `<description>` element is optional but can be quite useful for tracking details about the property it describes.

Your applications can specify additional resources, and they're also loaded in the order in which they're specified, *after* the system-defined resources have been loaded.

TIP Configuration parameter values can be declared as `final` so that user applications can't change them later, as shown in the following example (from a sample `hdfs-site.xml` file):

```
<property>
  <name>dfs.hosts.include</name>
  <value>/etc/hadoop/conf/hosts.include</value>
  <final>true</final>
</property>]]>
</line>
</lineatedText>
</computerCode>
```

After a value is declared to be `final`, no subsequently loaded resource can alter that value.

Finally, the following code example shows a configuration specification from a `mapred-site.xml` file:

```
<property>
 <name>mapred.local.dir</name>
 <value>/home/hadoop/hadoop-0.20.2/mapred-tmp</value>
 <description>Comma-separated list of paths on the local
            file system where temporary MapReduce data is
            written.</description>
</property>
```

A ton of parameter names and values are associated with the resources in the following list. To see a list of a resource's parameter names and values, check out its URL:

- `core-default.xml`: http://hadoop.apache.org/docs/current/hadoop-project- dist/hadoop-common/core-default.xml
- `hdfs-default.xml`: http://hadoop.apache.org/docs/current/hadoop-project- dist/hadoop-hdfs/hdfs-default.xml
- `mapred-default.xml`: http://hadoop.apache.org/docs/current/hadoop-mapreduce- client/hadoop-mapreduce-client-core/mapred-default.xml