Community Experience Distilled

Apache Accumulo for Developers

Build and integrate Accumulo clusters with various cloud platforms

Guðmundur Jón Halldórsson

[PACKT] open source community experience distilled PUBLISHING

Apache Accumulo for Developers

Build and integrate Accumulo clusters with various cloud platforms

Guðmundur Jón Halldórsson

[PACKT] open source
community experience distilled
PUBLISHING

BIRMINGHAM - MUMBAI

Apache Accumulo for Developers

First published: October 2013

Production Reference: 1101013

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-599-0

www.packtpub.com

Cover Image by Gant Man (gantman@gmail.com)

Credits

Author

Guðmundur Jón Halldórsson

Reviewers

Einar Th. Einarsson

Andrea Mostosi

Pálmi Skowronski

Acquisition Editor

Joanne Fitzpatrick

Commissioning Editor

Sharvari Tawde

Technical Editors

Aparna Kumari

Krutika Parab

Pramod Kumavat

Hardik B. Soni

Project Coordinator

Akash Poojary

Copy Editors

Brandt D'Mello

Gladson Monterio

Alfida Paiva

Proofreader

Simran Bhogal

Indexer

Rekha Nair

Graphics

Abhinash Sahu

Ronak Dhruv

Production Coordinator

Manu Joseph

Cover Work

Manu Joseph

About the Author

Guðmundur Jón Halldórsson is a Software Engineer who enjoys the challenges of complex problems and pays close attention to detail. He is an annual speaker at the Icelandic Computer Society (SKY, `http://www.utmessan.is/`).

Guðmundur is a Software Engineer with extensive experience and management skills, and works for Five Degrees (`www.fivedegrees.nl`), a banking software company. The company develops and sells high-quality banking software. As a Senior Software Engineer, he is responsible for the development of a backend banking system produced by the company. Guðmundur has a B.Sc. in Computer Sciences from the Reykjavik University.

Guðmundur has a long period of work experience as a Software Engineer since 1996. He has worked for a large bank in Iceland, an insurance company, and a large gaming company where he was in the core EVE Online team.

Guðmundur is passionate about whatever he does. He loves to play online chess and Sudoku. And when he has time, he likes to read science fiction and history books.

He maintains a Facebook page to network with his friends and readers, and blogs about the wonders of programming and cloud computing at `http://www.gudmundurjon.net/`.

I would like to thank my two girls, Kolbrún and Bryndís, for their patience while I was writing this book, and researching in the area of cluster computing.

About the Reviewers

Einar Th. Einarsson has been hacking computers since childhood, and has worked both as a Programmer and a System Administrator for more than 15 years in diverse fields such as online gaming, anti-malware, biotech, and telecommunications, at companies such as CCP Games, FRISK Software, and deCODE Genetics. He is currently the CTO of a startup company focused on providing tools for the online poker world.

Andrea Mostosi is a passionate Software Developer. In 2003, while he was at high school, he started with a single-node LAMP stack and grew up by adding more languages, components, and nodes. He graduated in Milan and worked on several web-related projects. He is currently working with data, trying to discover information hidden behind huge datasets.

I would like to thank my girlfriend Khadija, who lovingly supports me in everything I do, and the people I collaborated with, for fun or for work, for everything they taught me. I would also like to thank Packt Publishing and its staff for this opportunity to contribute to this production.

Pálmi Skowronski holds a bachelor's and a master's degree in Computer Science from Reykjavík University, with a focus on machine-learning and heuristic searches.

Most recently, he has been working in the financial sector developing distributed enterprise solutions with Five Degrees as a Senior Developer, and is currently working on smart analysis of financial transactions with Meniga as a Software Specialist.

I would like to thank the author Mr. Halldórsson, a friend and colleague, for the many laughs and stimulating conversations we had during the writing of this book. May there be many more in the near future.

---

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Apache Accumulo is a sorted, distributed Key-Value store. Since Accumulo depends on other systems, setting it up for the first time is slightly difficult, hence the aim of *Apache Accumulo for Developers* is to make this process easy for you by following a step-by-step approach. Monitoring, performance tuning, and optimizing an Accumulo cluster is difficult unless you have the right tools. This book shall take a deep dive into these tools and also address the security issues that come along with the Accumulo cluster.

What this book covers

Chapter 1, *Building an Accumulo Cluster from Scratch*, explores how to set up a single-node, pseudo-distributed mode and then expand it to a multi-node.

Chapter 2, *Monitoring and Managing Accumulo*, focuses on four major things to keep the cluster in a healthy state and to keep in check all the problems that occur while dealing with a cluster.

Chapter 3, *Integrating Accumulo into Various Cloud Platforms*, explores how to integrate Accumulo into various cloud platforms both as a single-node, pseudo-distributed mode and when it's expanded to a multi-node.

Chapter 4, *Optimizing Accumulo Performance*, focuses on how to optimize the performance of Accumulo. Since Accumulo uses Hadoop and ZooKeeper, we need to start off with performance optimization techniques for Hadoop and ZooKeeper before we go ahead with performance optimization for Accumolo.

Chapter 5, *Security*, reveals that Accumulo is designed for fine-grained security, which normal database systems do not support. Accumulo is designed to extend BigTable and supports full cell-level security.

Appendix A, *Accumulo Command References*, contains a list of all available commands in the Accumulo shell.

Appendix B, *Hadoop Command References*, contains a list of user commands and administrator commands in Hadoop.

Appendix C, *ZooKeeper Command References*, contains a list of ZooKeeper commands called "the four-letter words".

What you need for this book

Apache Accumulo for Developers will explain how to download and configure all the tools needed. This doesn't apply to the following tools, which you'll need to install beforehand:

- **Ganglia**: Ganglia is a scalable and distributed monitoring system for high-performance computing systems such as clusters and grids. See `http://ganglia.info` for more information.
- **Graylog2:** Graylog2 enables you to monitor application logs. See `http://graylog2.org` for more information.
- **Nagios**: Nagios is a powerful monitoring system. See `http://www.nagios.org` for more information.

Who this book is for

This book is designed for both developers and administrators, who will configure, administer, monitor, and even troubleshoot Accumulo. Both developers and administrators will gain an understanding of how to use Accumulo, the design of Accumulo, and learn about Accumulo's strength.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows:

The file `mapred-site.xml` can be used to configure the host and the port for the Map/Reduce JobTracker.

A block of code is set as follows:

```
String inName = "accumulo-demo";
String zooKeeperServers = "zkServer1,zkServer2,zkServer3";
Instance zkIn = new ZooKeeperInstance(inName, zooKeeperServers);
Connector conn = zkInstance.getConnector("myuser", "password");
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
String inName = "accumulo-demo";
String zooKeeperServers = "zkServer1,zkServer2,zkServer3";
Instance zkIn = new ZooKeeperInstance(inName, zooKeeperServers);
Connector conn = zkInstance.getConnector("myuser", "password");
```

Any command-line input or output is written as follows:

```
root@accumulo-demo mydemotable3> scan -s SecTokenB
2013-08-19 23:45:24,709 [shell.Shell] ERROR:
   java.lang.RuntimeException:
   org.apache.accumulo.core.client.AccumuloSecurityException:
   Error BAD_AUTHORIZATIONS - The user does not have the specified
   authorizations assigned
```

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Building an Accumulo Cluster from Scratch

Apache Accumulo was created in 2008 by National Security Agency (NSA), and contributed to the Apache Foundation in 2011. Accumulo is a sorted, distributed Key-Value store based on Google's BigTable design and high performance data store and retrieval system. Accumulo uses Apache Hadoop HDFS for storage, ZooKeeper for coordination, and Thrift. Apache Thrift software framework is used to define and create services for many languages such as C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, and many others. Thrift will not be discussed in this book, but is worth looking at.

There are few prerequisites for deploying Accumulo; the ZooKeeper cluster needs to be up and running and HDFS needs to be configured and running before Accumulo can be initialized.

In this chapter, we will explore how to set up a single-node, pseudo-distributed mode and then expand it to a multi-node. In a multi-node scenario, placing the machines in odd numbers is the best setup because ZooKeeper requires a majority. For example, with five machines, ZooKeeper can handle the failure of two machines; with three machines, ZooKeeper can handle the failure of one machine. As Accumulo depends on other systems, it can be hard to set it up for the first time.

This chapter will give you an answer to the question of how to set up Accumulo.

These are the topics we'll cover in this chapter:

- Necessary requirements
- Setting up Cygwin
- Setting up Hadoop (Version 1.2.1)
- Setting up ZooKeeper (Version 3.3.6)
- Setting up and configuring Accumulo (Version 1.4.4)
- Starting the Accumulo cluster
- Connecting to the Accumulo cluster using Java

Necessary requirements

When setting up Accumulo for development purposes, hardware requirements are usually not the issue; you just make do with what you have, but having more memory and a good CPU is always helpful. In most cases, a Map/Reduce job will encounter a bottleneck in two scenarios:

- I/O-bound job when reading data from a disk
- CPU-bound job when processing data

More information about Map/Reduce can be found at the following link:

```
http://en.wikipedia.org/wiki/Map_Reduce
```

There is big difference when setting up Apache Hadoop, ZooKeeper, or Accumulo on Windows or Linux. To make the difference less visible, all examples on Windows will use Cygwin, and in some cases Windows PowerShell. All examples using Windows PowerShell need administrator privileges. IPv6 should be disabled on both Linux and Windows machines to minimize the risk of Hadoop binding to the IPv6 address (I have seen this on Ubuntu machines).

To disable IPv6 for Linux, add or change the following lines in the `sysctl.conf` file in the `etc` directory:

```
# Disable IPv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

To disable IPv6 for Windows:

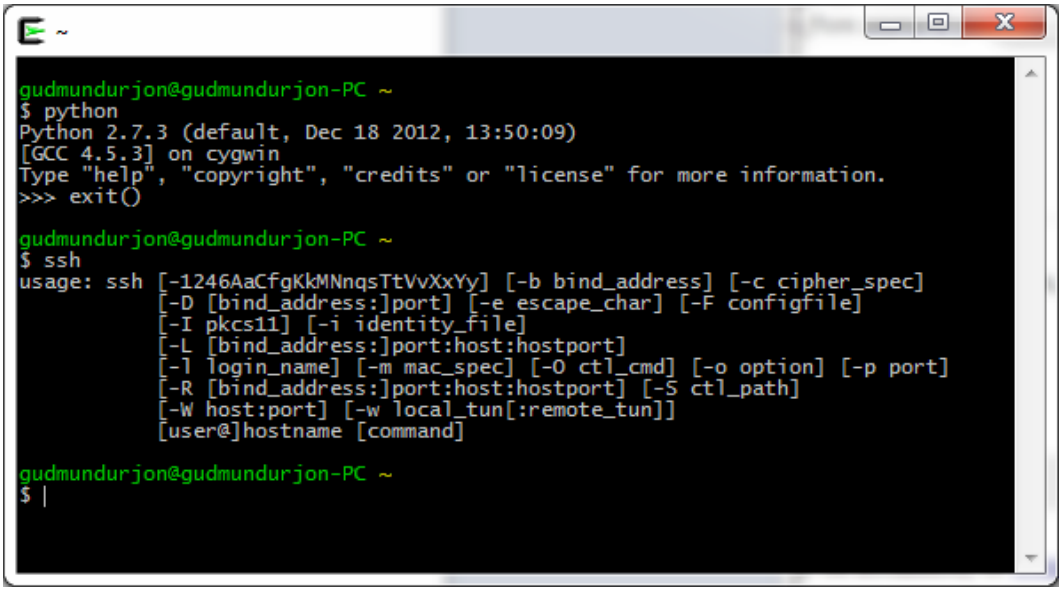
1. Open `RegEdit.exe`.
2. Navigate to `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TCPIP6\Parameters`.
3. Right click on the white background and add a new **DWORD (32-bit) Value**, and then edit it with the value `0`.

Setting up Cygwin

Many examples in this book use Cygwin. Cygwin is a set of tools that provide a Linux flavor for Windows. It is very important to know that Cygwin isn't a way to run native Linux applications on Windows. Download Cygwin (32-bit) from `http://cygwin.com/setup-x86.exe` and run. Pick the following packages:

- `openssh`: The OpenSSH server and its client programs
- `openssl`: The OpenSSL base environment
- `wget`: Utility to retrieve files from WWW via HTTP and FTP
- `python`: Python language interpreter
- `nano`: Enhanced clone of the Pico editor
- `vim`: Vi IMproved—enhanced vi editor

After installing Cygwin, open up the Cygwin Terminal and try to run the command `python`, and then the command `ssh` to verify whether the setup has been executed correctly. The Cygwin window should look as follows:



Setting up Hadoop

Hadoop is a Java application framework and is designed to run on a large cluster of inexpensive hardware. As Hadoop is written in Java, it requires a working Java 1.6.x installation. Both SSH and SSHD must be running to use the Hadoop scripts remotely. For Windows installation, Cygwin is required. If Hadoop is already installed and running, you can skip this section.

SSH configuration

Hadoop uses SSH access to manage its nodes, both remote and local machines. Even if we only want to set up a local development box, we need to configure SSH access. To simplify, we should create a dedicated Hadoop user (we are going to do this for ZooKeeper and Accumulo in later sections of this chapter).

Creating a Hadoop user

A Hadoop user can be created in different ways in Linux and Windows.

To create a Hadoop user for Linux, enter the following command-line code:

```
sudo addgroup hadoopgroup
sudo adduser –ingroup hadoopgroup hadoopuser
```

We want to isolate Hadoop by creating a dedicated Hadoop user account for running Hadoop. We are doing this because everything is running on the same machine in the beginning, and in most cases, this is going to be your developer machine.

To create a Hadoop user for Windows (PowerShell), use the Windows `net` command to perform the same steps as in Linux:

```
net localgroup "hadoopgroup" /add
net user "hadoopuser" "!Qwert1#" /add
net localgroup hadoopgroup "hadoopuser" /add
```

Generating an SSH key for the Hadoop user

An SSH key for the Hadoop user can be generated in different ways in Linux and Windows.

To generate an SSH key for Linux (remember SSH has to be installed), enter the following command-line code:

```
su - hadoopuser
ssh-keygen –t rsa -P ""
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
ssh hadoopuser@localhost
```

Run the shell with the substitute user, `hadoopuser`, to create a new `rsa` key, and change the passphrase of the private key to an empty string; otherwise, Hadoop will request the passphrase every time it interacts with its nodes. When this has been done, we need to enable SSH access to your local machine by copying the `id_rsa.pub` file to the `authorized_keys` directory.

To generate an SSH key for Windows, enter the following command-line code in the Cygwin Terminal (with administrator privileges, else you will have to enter administrator/system password whenever asked):

```
ssh-host-config -y
cygrunsrv –S sshd
```

For every **(yes/no)** question in the `ssh-host-config` script, the default answer is **yes**. Then, start the `sshd` service by using the Cygwin `cygrunsrv` command.

Installing Hadoop

Hadoop has NameNode, JobTracker, DataNode, and TaskTracker:

- **NameNode**: It keeps the directory tree of all files in the filesystem, and tracks where across the cluster, the datafile is kept. It does not store the data of these files itself.
- **JobTracker**: It is the service within Hadoop that farms out Map/Reduce tasks to specific nodes in the cluster—ideally the nodes that have data, or nodes that are at least in the same rack.
- **DataNode**: It stores data in the Hadoop filesystem (discussed later in this chapter).
- **TaskTracker**: It is a node in the cluster that accepts tasks.

Installation of the Hadoop cluster typically involves unpacking the software on all the machines in the cluster. In a multi-node setup:

- The first machine is designated as the `NameNode`
- The second machine is designated as the `JobTracker`
- The third machine acts as both `DataNode` and `TaskTracker` and are the slaves

Multi-node clusters will be discussed later in this chapter. The rule of thumb is to create single-node setups, and then change the configuration files in order to change a single-node setup into a multi-node setup.

For installing Hadoop on Linux, enter the following command-line code:

```
cd /usr/local
sudo wget http://apache.mirrors.tds.net/hadoop/common/hadoop-1.2.1/
hadoop-1.2.1.tar.gz
sudo tar xzf hadoop-1.2.1.tar.gz
sudo mv hadoop-1.2.1 hadoop
sudo chown -R hadoopuser:hadoopgroup hadoop
```

Use `wget` to download the Hadoop version we want to set up. Currently, 1.2.1 is the stable version, but please check this before continuing and update if needed. After getting the file, we need to extract it. Instead of using the default name, we have two options: one is to rename it as we are doing here, and the other is to use `symlink` (this is easier when we update the Hadoop node). Finally, recursively change the ownership of the given directory to the Hadoop user.

For installing Hadoop on Windows, there are two options. The first one is to use WebClient in the .NET framework to download the file to the same location used in the example in the preceding Linux section. This can be done using Windows PowerShell (with administrator privileges). Enter the following command-line code in Windows PowerShell:

```
$url = "http://apache.mirrors.tds.net/hadoop/common/hadoop-1.2.1/hadoop-1.2.1.tar.gz"
$dir = "c:\cygwin\usr\local"
$webclient = New-Object System.Net.WebClient
$webclient.DownloadFile($url, "$dir\hadoop-1.2.1.tar.gz")
```

The second option is to use Cygwin Terminal (with administrator privileges). Enter the following command-line code in the Cygwin Terminal:

```
cd /usr/local
wget http://apache.mirrors.tds.net/hadoop/common/hadoop-1.2.1/hadoop-1.2.1.tar.gz
tar xzf hadoop-1.2.1.tar.gz
mv hadoop-1.2.1 hadoop
```

For consistency, use Cygwin's `mv` command as in the Linux example.

Configuring Hadoop

The Hadoop configuration is driven by two types of important configuration files, which need to be configured for Hadoop to run as expected.

- **Read-only default configuration**: Files for this configuration are `core-default.xml`, `hdfs-default.xml`, and `mapred-default.xml`
- **Site-specific configuration**: Files for this configuration are `core-site.xml`, `hdfs-site.xml`, and `mapred-site.xml`

All Hadoop configuration files are located at `/usr/local/hadoop/conf` on Linux, or at `C:\cygwin\usr\local\hadoop\conf` on Windows The default files present at the given location are listed as follows:.

- `capacity-scheduler.xml`: This is the configuration file for the resource manager in Hadoop. It is used for configuring various scheduling parameters related to queues.
- `configuration.xsl`: This is an extensible stylesheet language file used for the `hadoop-site.xml` file.
- `core-site.xml`: This is a site-specific file used to override default values of core Hadoop properties.
- `fair-scheduler.xml`: This file contains the pool and user allocations for the Fair Scheduler. For more information, please go to `http://hadoop.apache.org`.
- `hadoop-env.sh`: Using this file, we can set Hadoop-specific environment variables. The only required environment variable is `JAVA_HOME`.
- `hadoop-metrics2.properties`: Using this file, we can set up how Hadoop is monitored.
- `hadoop-policy.xml`: This is a site-specific file used to override default policies, such as access control properties of Hadoop. It is used to configure ACL for `ClientProtocol`, `ClientDatanodeProtocol`, `JobSubmissionProtocol`, `TaskUmbilicalProtocol`, and `AdminOperationsProtocol`.
- `hdfs-site.xml`: This is a site-specific file used to override default properties of Hadoop filesystem.
- `log4j.properties`: Using this file, we can configure appenders for:
    - Job Summary
    - Daily Rolling File
    - 30-day backup
    - TaskLog
    - Security audit
    - Event Counter
- `mapred-queue-acls.xml`: This file contains the access control list for user and group names that are allowed to submit jobs. Alternatively, it contains user and group names that are allowed to view job details, kill jobs, or modify job's priority for all the jobs.

- `mapred-site.xml`: This is a site-specific file used to override default values for Hadoop Map/Reduce properties.
- `masters`: This is the master hostname file.
- `slaves`: This is the slaves hostname file.
- `ssl-client-xml.example`: This is an example file that ships with Hadoop. There is no need to change this file.
- `ssl.server.xml.example`: Also, an example file that ships with Hadoop. There is no need to change this file.
- `taskcontroller.cfg`: There is no need to change this file.

To get your single node up and running, we only need to change three files: `core-site.xml`, `hdfs-site.xml`, and `mapred-site.xml`.

core-site.xml

Replace the code in `core-site.xml` with the following code:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>Hadoop temp dir</description>
  </property>

  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:54310</value>
    <description>Name and location of default filesystem
    </description>
  </property>
</configuration>
```

Hadoop needs a directory for temporary files. Make sure you are in the root directory. Enter `cd` in command line (either Linux or Windows Cygwin Terminal) and press *Enter* to be on the safer side.

For Linux, create the directory for the Hadoop filesystem and configure access for it. Enter the following command-line code:

```
sudo mkdir -p /app/hadoop/tmp
sudo chown hadoopuser:hadoopgroup /app/hadoop/tmp
```

For Windows, create the directory for the Hadoop filesystem. Enter the following command-line code in the Cygwin Terminal:

```
mkdir -p /app/hadoop/tmp
```

There is no need to worry that much about security at this point, but it's a good practice to secure the directory as much as possible.

mapred-site.xml

The file `mapred-site.xml` can be used to configure the host and the port for the Map/Reduce JobTracker. Replace the code in `mapred-site.xml` with the following code:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:54311</value>
    <description>The Map/Reduce job tracker
    </description>
  </property>
</configuration>
```

hdfs-site.xml

The file `hdfs-site.xml` can be used to specify the actual number of replications when the file `dfs.replication` is created. Replace the code in `hdfs-site.xml` with the following code:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default replication
    </description>
  </property>
</configuration>
```

hadoop-env.sh

To be able to start Hadoop, it is important to change the `hadoop-env.sh` file and uncomment the `JAVA_HOME` line; also, point to the correct instance of JVM to let Hadoop know where Java is located. Please note that this procedure applies to Linux and not Windows.

For Windows, click on the **Start** button, then right-click on **My Computer** (XP) or **Computer** (Win Vista/7), go to **Properties**, then go to **Advanced** tab (XP) or **Advanced system settings** (Win Vista/7), and then click on the **Environment Variables...** button. There, add a new user variable with **Variable name** as `JAVA_HOME` and with **Variable value** as your Java path, for example, `C:\Program Files\Java\jdk1.7.0_09`. Also, below the **User variables**, in **System variables**, edit the **Path** variable and append the following line in it: `C:\cygwin\bin;C:\cygwin\usr\sbin`.

Preparing the Hadoop filesystem

Before starting Hadoop for the first time, it is required to format the Hadoop filesystem (often called **Hadoop Distributed File System** (**HDFS**) or just **Hadoop DFS**, which is designed to store very large files across machines in a large cluster). And remember, if you format a running Hadoop filesystem, all your data will be deleted. Keeping files under `/app/<application name>/tmp` instead of `/tmp/<application name>` is good practice.

For Linux, format the Hadoop nodes as `hadoopuser`; this is required because of access restrictions. Enter the following command-line code:

```
su - hadoopuser
/usr/local/hadoop/bin/hadoop namenode -format
```

For Windows, enter the following command-line code in the Cygwin Terminal:

```
/usr/local/hadoop/bin/hadoop namenode -format
```

If you get some errors in Cygwin, something like
/usr/local/hadoop/bin/hadoop: line 2: $'\r': command not found.
Then, enter the following command-line code and try formatting again:

```
dos2unix /usr/local/hadoop/bin/hadoop
```

Starting the Hadoop cluster

The `bin` directory located at `/usr/local/hadoop` contains few useful scripts to start or stop Hadoop DFS and the Hadoop Map/Reduce daemons:

- `start-dfs.sh`: This script starts the Hadoop DFS daemons (NameNode and DataNode)
- `stop-dfs.sh`: This script stops the Hadoop DFS daemons
- `start-mapred.sh`: This script starts the Hadoop Map/Reduce daemons (JobTracker and TaskTracker)
- `stop-mapred.sh`: This script stops the Hadoop Map/Reduce daemons
- `start-all.sh`: This script starts all of the Hadoop daemons (NameNode, DataNode, JobTracker, and TaskTracker)
- `stop-all.sh`: This script stops all of the Hadoop daemons

Use `start-all.sh` to start all of the Hadoop daemons (either Linux or Windows Cygwin Terminal) as follows (make sure you have administrator rights, or you will have to enter the administrator/system password whenever asked for):

```
/usr/local/hadoop/bin/start-all.sh
```

To confirm that the processes are launched, view the web interface for NameNode at `http://localhost:50070`, the JobTracker at `http://localhost:50030`, and finally the TaskTracker at `http://localhost:50060`.

Multi-node configurations

For setting up multiple nodes, it is good practice to use names instead of IP addresses. In this example, let's name the nodes `masternode` (10.0.0.1), `slavenode1` (10.0.0.2), and `slavenode2` (10.0.0.3). It is required to set up all the nodes in the same way.

Update the `hosts` file located at `/etc` on all the nodes, as follows:

```
10.0.0.1 masternode
10.0.0.2 slavenode1
10.0.0.3 slavenode2
```

Next, distribute the SSH public key of `hadoopuser@masternode` to `slavenode1` and `slavenode2`. This step can be done manually or by using the `ssh-copy-id` script from the masternode machine, as follows:

```
ssh-copy-id -i $HOME/.ssh/id_rsa.pub hadoopuser@slavenode1
ssh-copy-id -i $HOME/.ssh/id_rsa.pub hadoopuser@slavenode2
ssh masternode
ssh slavenode1
ssh slavenode2
```

You need to connect to both of the nodes to permanently add RSA (`masternode`, `slavenode1`, and `slavenode2`), to the list of known hosts.

On the masternode machine, you need to edit the file `masters` located in `/usr/local/hadoop/conf` by typing in `masternode`, and edit the file `slaves` located in `/usr/local/hadoop/conf` by typing in the names of the nodes as follows:

```
masternode
slavenode1
slavenode2
```

On all of the machines, you need to change the following files:

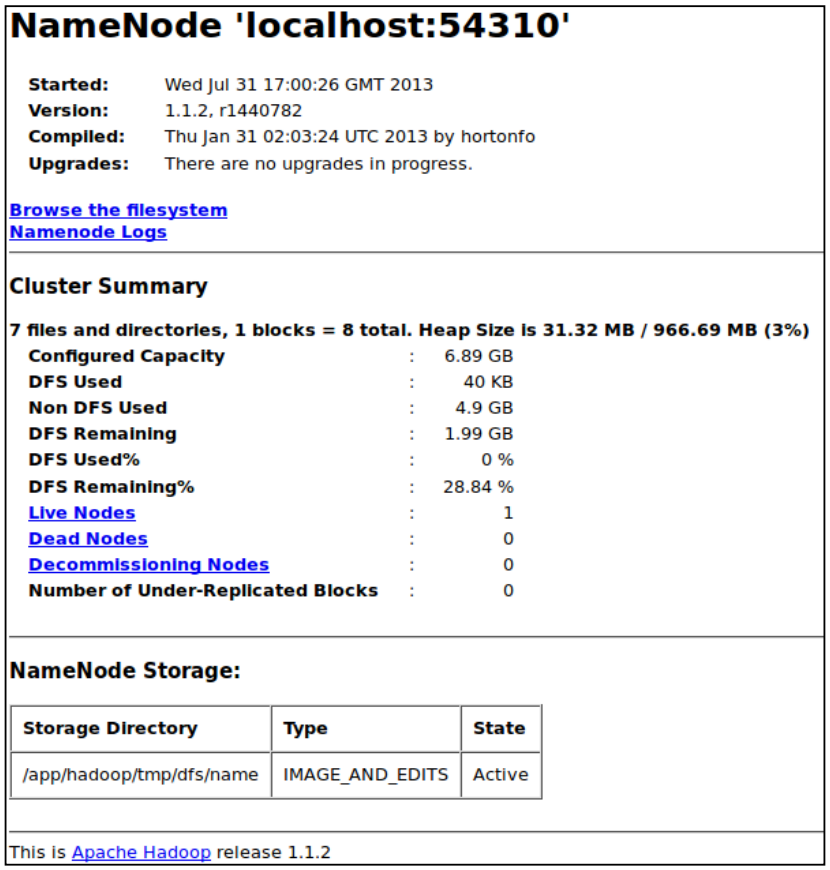
- `core-site.xml` located at `/usr/local/hadoop/conf` should be changed from `<value>localhost:54310</value>`, to `<value>masternode:54310</value>`
- `mapred-site.xml` located at `/usr/local/hadoop/conf` should be changed from `<value>localhost:54311</value>`, to `<value>masternode:54311</value>`

To start the nodes, run the following command (either Linux or Windows Cygwin Terminal) on the masternode machine:

```
/usr/local/hadoop/bin/start-dfs.sh
```

The NameNode website

To verify that the NameNode is up and running as intended, browse to `http://localhost:54310`. The information displayed on this page is very important to see the status of a single- or multi-cluster setup, as shown in the following screenshot:



NameNode 'localhost:54310'

Started: Wed Jul 31 17:00:26 GMT 2013
Version: 1.1.2, r1440782
Compiled: Thu Jan 31 02:03:24 UTC 2013 by hortonfo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

7 files and directories, 1 blocks = 8 total. Heap Size is 31.32 MB / 966.69 MB (3%)

Configured Capacity	:	6.89 GB
DFS Used	:	40 KB
Non DFS Used	:	4.9 GB
DFS Remaining	:	1.99 GB
DFS Used%	:	0 %
DFS Remaining%	:	28.84 %
Live Nodes	:	1
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

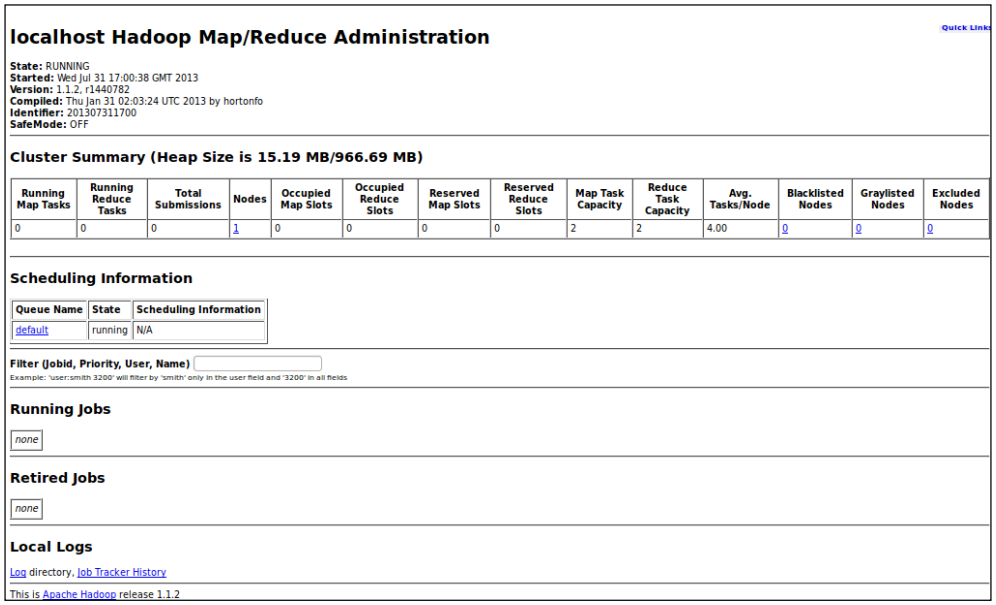
NameNode Storage:

Storage Directory	Type	State
/app/hadoop/tmp/dfs/name	IMAGE_AND_EDITS	Active

This is Apache Hadoop release 1.1.2

The JobTracker website

The JobTracker website displays an overview of the general job statistics running on the Hadoop cluster. It also displays completed or failed jobs, as shown in the following screenshot:



localhost Hadoop Map/Reduce Administration

Quick Links

State: RUNNING
Started: Wed Jul 31 17:00:38 GMT 2013
Version: 1.1.2, r1440782
Compiled: Thu Jan 31 02:03:24 UTC 2013 by hortonfo
Identifier: 201307311700
SafeMode: OFF

Cluster Summary (Heap Size is 15.19 MB/966.69 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	0	1	0	0	0	0	2	2	4.00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Retired Jobs

none

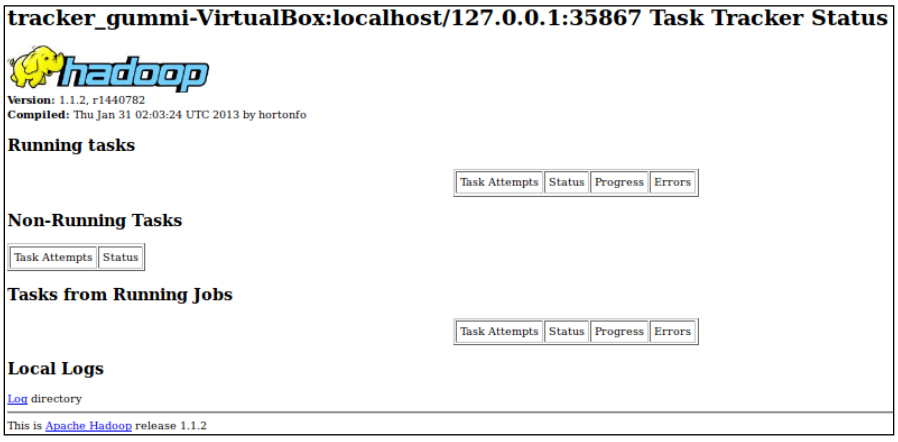
Local Logs

Log directory, Job Tracker History

This is Apache Hadoop release 1.1.2

The TaskTracker website

The TaskTracker website displays an overview of the running and nonrunning tasks, and the log directory, as shown in the following screenshot:



tracker_gummi-VirtualBox:localhost/127.0.0.1:35867 Task Tracker Status

Version: 1.1.2, r1440782
Compiled: Thu Jan 31 02:03:24 UTC 2013 by hortonfo

Running tasks

Task Attempts	Status	Progress	Errors

Non-Running Tasks

Task Attempts	Status

Tasks from Running Jobs

Task Attempts	Status	Progress	Errors

Local Logs

Log directory

This is Apache Hadoop release 1.1.2

Setting up ZooKeeper

ZooKeeper is designed to be a high performance coordination service for the cluster environment running distributed applications. By allowing distributed processes to coordinate using a simple namespace (similar to a filesystem) stored in memory, a high throughput and low latency is achieved. The best performance is reached when read versus write ratios are around 10:1.

ZooKeeper is written in Java and therefore requires a working Java 1.6.x installation. For Windows installation, Cygwin is required.

Installing ZooKeeper

When ZooKeeper is set up, every node has to know about each other. ZooKeeper follows the concept of "follow the leader", which means there is always a leader machine, and the leader is chosen when the ZooKeeper cluster starts up. Clients only connect to the nodes that follow the leader, not the leader node.

ZooKeeper guarantees the following:

- **Sequential Consistency**: Updates from a client will be applied in the order that they were sent
- **Atomicity**: Updates either succeed or fail; there are no partial results.
- **Single System Image**: A client will see the same view of the service regardless of the server that it connects to
- **Reliability**: Once an update has been applied, it will persist from that time onward until a client overwrites the update
- **Timeliness**: The client's view of the system is guaranteed to be up-to-date within a certain time bound

For Linux, use `wget` to download the ZooKeeper version that we want to set up. Stable versions are 3.3.6 and 3.4.5. Please check this for the stable version before continuing and update if needed. After getting the file, we need to extract it. Instead of using the default name, we have two options: one is to rename it as we are doing here, and the other is to use `symlink` (this is easier when we update the ZooKeeper node). Finally, recursively change the ownership of the given directory to the Hadoop user (you can create a separate user for ZooKeeper).

Enter the following command-line code in Linux:

```
cd /usr/local
sudo wget http://apache.mirrors.tds.net/zookeeper/zookeeper-3.3.6/
zookeeper-3.3.6.tar.gz
sudo tar xzf zookeeper-3.3.6.tar.gz
sudo mv zookeeper-3.3.6 zookeeper
sudo chown -R hadoopuser:hadoopgroup zookeeper
```

For Windows, there are two options. The first option is to use WebClient in the .NET framework to download the file to the same location used in the example in the preceding Linux section. This can be done using Windows PowerShell (with administrator privileges). Enter the following command-line code in Windows PowerShell:

```
$url = "http://www.globalish.com/am/zookeeper/zookeeper-3.3.5/zookeeper-
3.3.5.tar.gz"
$dir = "c:\cygwin\usr\local"
$webclient = New-Object System.Net.WebClient
$webclient.DownloadFile($url, "$dir\zookeeper-3.3.5.tar.gz")
```

The second option is to use Cygwin Terminal (with administrator privileges). Enter the following command-line code in the Cygwin Terminal:

```
cd /usr/local
wget http://apache.mirrors.tds.net/zookeeper/zookeeper-3.3.6/zookeeper-
3.3.6.tar.gz
tar xzf zookeeper-3.3.6.tar.gz
mv zookeeper-3.3.6 zookeeper
```

Configuring ZooKeeper

ZooKeeper needs one configuration file `zoo.cfg` in the `conf` directory, which is created with the following script:

```
cat /usr/local/zookeeper/conf/zoo_sample.cfg >>
 /usr/local/zookeeper/conf/zoo.cfg
```

Add the following properties into the newly created `zoo.cfg` file:

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
```

```
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# The directory where the snapshot is stored
dataDir=/app/zookeeper
# the port which the client will connect
clientPort=2181
```

For a multi-node setup, you need to add a few lines to the `zoo.cfg` file. In the following example, three nodes have been created with the names `zookeeper1` (10.0.0.10), `zookeeper2` (10.0.0.11), and `zookeeper3` (10.0.0.12). Adding these names to the host file is required for every node, as we saw in the multi-node configuration for Hadoop:

```
# Cluster
server.1=zookeeper1:2888:3888
server.2=zookeeper2:2888:3888
server.3=zookeeper3:2888:3888
```

For more information about the ZooKeeper cluster setup, visit `http://zookeeper.apache.org/doc/r3.3.6/zookeeperAdmin.html#sc_zkMulitServerSetup`

ZooKeeper needs a directory for temporary files. Make sure you are in the root directory. Enter `cd` in command line (either Linux or Windows Cygwin Terminal) and press *Enter* to be on the safer side. We will now create and give access to a Hadoop user on Linux.

For Linux, create the directory for the ZooKeeper filesystem. To do this, enter the following command-line code:

```
sudo mkdir -p /app/zookeeper
sudo chown hadoopuser:hadoopgroup /app/zookeeper
```

For Windows, enter the following command-line code in the Cygwin Terminal:

```
mkdir -p /app/zookeeper
```

Starting ZooKeeper

Start ZooKeeper with the following command-line code (either Linux or Windows Cygwin Terminal):

```
/usr/local/zookeeper/bin/zkServer.sh start
```

To verify that ZooKeeper is running as it should be, enter the following command-line code (either Linux or Windows Cygwin Terminal):

```
/usr/local/zookeeper/bin/zkCli.sh
```

For more information about ZooKeeper commands, see *Appendix C*.

Setting up and configuring Accumulo

The last step in the setup process of Accumulo is to hook Accumulo to Hadoop and ZooKeeper, that we configured in the previous sections.

Installing Accumulo

Use `wget` to download the Accumulo version we want to set up. Currently, 1.4.2 is the latest version, but please check this before continuing and update if needed. After getting the file, we need to extract it. Instead of using the default name we have two options: one is to rename it as we are doing here, and the other is to use `symlink` (that is easier when we update the Accumulo node). Finally, recursively change the ownership of the given directory to our Hadoop user; `hadoopuser` (you can create a separate user for Accumulo).

For Linux, enter the following command-line code:

```
cd /usr/local
sudo wget http://apache.mirrors.tds.net/accumulo/1.4.4/accumulo-1.4.4-dist.tar.gz
sudo tar xzf accumulo-1.4.4-dist.tar.gz
sudo mv accumulo-1.4.4 accumulo
sudo chown -R hadoopuser:hadoopgroup accumulo
```

For Windows, there are two options. The first option is to use WebClient in the .NET framework to download the file to the same location as in the example in the preceding Linux section. This can be done using Windows PowerShell (with administrator privileges). Enter the following command-line code in Windows PowerShell:

```
$url = "http://apache.mirrors.tds.net/accumulo/1.4.4/accumulo-1.4.4-dist.tar.gz"
$dir = "c:\cygwin\usr\local"
$webclient = New-Object System.Net.WebClient
$webclient.DownloadFile($url, "$dir\accumulo-1.4.4-dist.tar.gz")
```

The second option is to use Cygwin Terminal (with administrator privileges). Enter the following command-line code in Cygwin Terminal:

```
cd /usr/local
wget http://apache.mirrors.tds.net/accumulo/1.4.4/accumulo-1.4.4-dist.
tar.gz
tar xzf accumulo-1.4.4-dist.tar.gz
mv accumulo-1.4.4 accumulo
```

Configuring Accumulo

For configuring Accumulo in a development environment like ours, we are going to use a small instance of Accumulo (512 MB).

Accumulo comes with example configuration files that are present in the `examples` directory located in `/usr/local/accumulo/conf`. There are examples for 512 MB, 1 GB, 2 GB, and 3 GB. In a local development scenario, there is no need to use an instance larger than 512 MB. To do this, change to the `hadoopuser` mode and copy all of the example configuration files for 512 MB standalone machine to the `conf` directory.

For Linux, enter the following command-line code:

```
su - hadoopuser
cp /usr/local/accumulo/conf/examples/512MB/standalone/*
 /usr/local/accumulo/conf
```

For Windows, enter the following command-line code in Cygwin Terminal:

```
cp /usr/local/accumulo/conf/examples/512MB/standalone/*
 /usr/local/accumulo/conf
```

The final step is to edit the `accumulo-env.sh` file located in `/usr/local/accumulo/conf`, and set your `JAVA_HOME`, `HADOOP_HOME`, and `ZOOKEEPER_HOME` as we did earlier for the `hadoop-env.sh` file while configuring Hadoop.

Starting the Accumulo cluster

Before starting Accumulo for the first time, initializing is required by using the `accumulo init` command to create the HDFS directory structure and ZooKeeper settings.

Initialize Accumulo with the following command (either Linux or Windows Cygwin Terminal), and name the resulting instance `accumulo-demo`, and choose a password for the root:

```
/usr/local/accumulo/bin/accumulo init
```

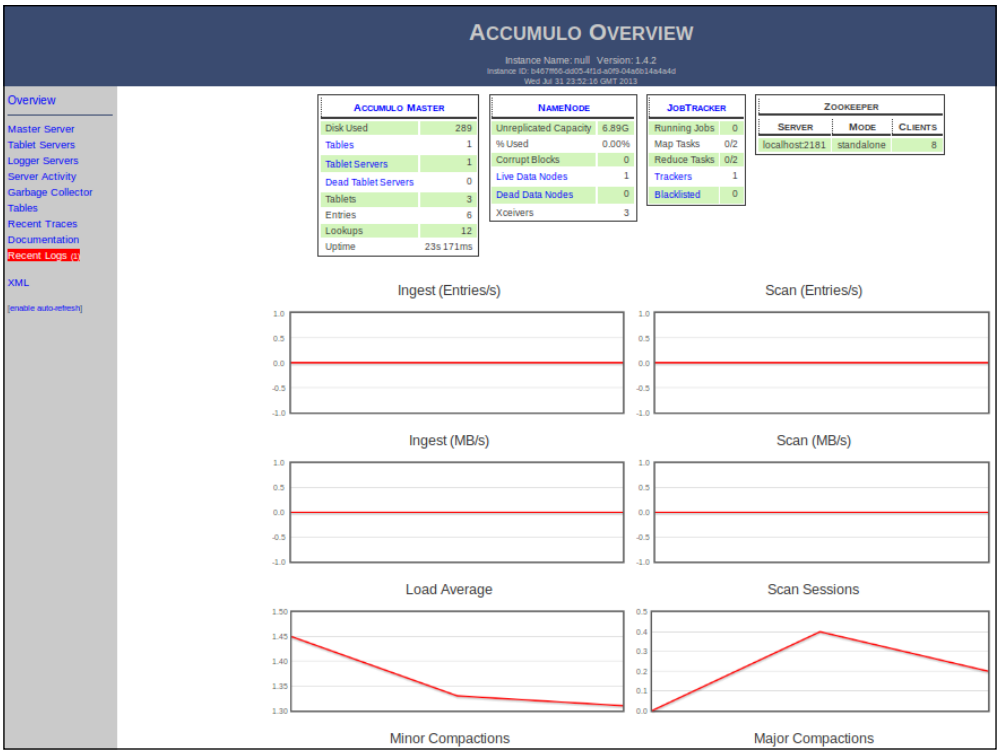
Start Accumulo with the following command (either Linux or Windows Cygwin Terminal):

```
/usr/local/accumulo/bin/start-all.sh
```

To confirm that the processes have been launched, view the web interface for Accumulo at `http://localhost:50095`.

The Accumulo website

The Accumulo website gives an overview of the **Ingest** and **Scan** entries among other metrics. Notice the highlighted menu item on the left, informing there are items in the log worth looking at. Often it is good rule to have the Accumulo website open with auto-refresh enabled when working with Accumulo. Learning how Accumulo handles different workloads helps when the performance tuning starts.



Connecting to the Accumulo cluster using Java

After setting up Accumulo, the first task that we need to do is create a simple application that connects to the development environment and writes to a table. There are two `.jar` files needed for this demo to work. The first file, `accumulo-core.1.4.4.jar`, is located at `/usr/local/accumulo/lib` on Linux, or `C:\cygwin\usr\local\accumulo\lib` on Windows. The second file, `hadoop-core.1.2.1.jar`, is located at `/usr/local/hadoop` on Linux, or `C:\cygwin\usr\local\hadoop` on Windows.

For our application, create a new Java project and add the two jar files in it. Then create a new Java class named `Accumulo1` and copy the following code in it. Do not forget to enter values of `userName` and `password` variables. Execute the Java code. It is preferable to use a dedicated IDE like Eclipse.

```
import org.apache.accumulo.core.client.AccumuloException;
import org.apache.accumulo.core.client.AccumuloSecurityException;
import org.apache.accumulo.core.client.BatchWriter;
import org.apache.accumulo.core.client.Connector;
import org.apache.accumulo.core.client.Instance;
import org.apache.accumulo.core.client.TableExistsException;
import org.apache.accumulo.core.client.TableNotFoundException;
import org.apache.accumulo.core.client.ZooKeeperInstance;
import org.apache.accumulo.core.data.Mutation;
import org.apache.accumulo.core.data.Value;
import org.apache.accumulo.core.security.ColumnVisibility;
import org.apache.hadoop.io.Text;

public class AccumuloDemo1 {
    public static void main(String[] args)throws
 AccumuloException, AccumuloSecurityException,
 TableNotFoundException, TableExistsException {

        // Constants
        String instanceName = "accumulo-demo";
        String zooServers = "zooList";
        String userName = "<change>";
        String password = "<change>";

        // Connect
        Instance inst = new ZooKeeperInstance(instanceName,
 zooServers);
        Connector conn = inst.getConnector(userName, password);

        // Use batch writer to write demo data
        BatchWriter bw = conn.createBatchWriter("demotable",
```

```
 1000000, 60000, 2);

        // set values
        Text rowID = new Text("row1");
        Text colFam = new Text("colFam");
        Text colQual = new Text("colQual");

        // set visibility
        ColumnVisibility colVis = new ColumnVisibility("public");
        long timestamp = System.currentTimeMillis();

        // set value
        Value value = new Value("some-value".getBytes());

        // create new mutation and add rowID, colFam, colQual, and
value
        Mutation mutation = new Mutation(rowID);
        mutation.put(colFam, colQual, colVis, timestamp, value);

        // add the mutation to the batch writer
        bw.addMutation(mutation);

        // close the batch writer
        bw.close();
    }
}
```

Summary

The setup phase for Accumulo is not simple in itself because it requires two other applications to be up and running prior to Accumulo. But this chapter showed you the easy-to-follow steps that set up and scale Hadoop, ZooKeeper, and Accumulo. And finally, this chapter showed you how to write a simple client program that connects to Accumulo and writes data to it. Accumulo also supports server-side coding using Map/Reduce.

The next step is to start solving problems using Accumulo. In the next chapter, the focus will shift to management of the Accumulo cluster, and how to spot problems when they happen in our cluster. We will focus on how to keep the cluster in a healthy state, and how to find problems that can occur, as quickly as possible.

2
Monitoring and Managing Accumulo

In the previous chapter, we went through how to set up a single- and multi-node cluster for three different applications: Hadoop, ZooKeeper, and Accumulo. In this chapter, we will focus on how to keep the cluster in a healthy state and how to find problems that can occur as quickly as possible.

There are a few things that we need to monitor:

- **Performance**: By monitoring the performance of our machine(s), we can immediately detect performance issues and analyze performance trends over time.
- **Process**: Monitor the processes of the machine(s) to make sure that one of the specified jobs or processes are running as intended, and also monitor the memory consumption of processes to detect memory leaks (more advanced).
- **Application**: Hadoop, ZooKeeper, and Accumulo produce logs that we want to monitor.
- **Uptime**: Monitor the uptime for the nodes in our cluster and maintain the history.
- **Dashboard**: It is very important to view the status of your entire cluster. Thus, creating dashboards or using the existing dashboards to get a spot problem will give you an even better value. Dashboards should focus on problems and notify you in a clear way.

To accomplish the process of monitoring and displaying results on dashboards, we are going to use three applications:

- **Ganglia**: This is a scalable, distributed monitoring system for high-performance computing systems such as clusters and grids.
- **Graylog2**: This enables you to monitor application logs. It is an open source log management solution that stores your logs in **ElasticSearch**. The messages are accepted via TCP, UDP, or even AMQP, and are stored in MongoDB.
- **Nagios**: This is a powerful monitoring system that enables organizations to identify and resolve IT infrastructure problems before they affect critical business processes.

  Nagios is not required, but is a standard monitoring system for many companies.

For more information about these applications, refer to the following links:

- **Ganglia**: `http://ganglia.info`
- **Graylog2 server**: `http://graylog2.org`
- **Nagios**: `http://www.nagios.org`

Specifically, we'll be covering the following topics in this chapter:

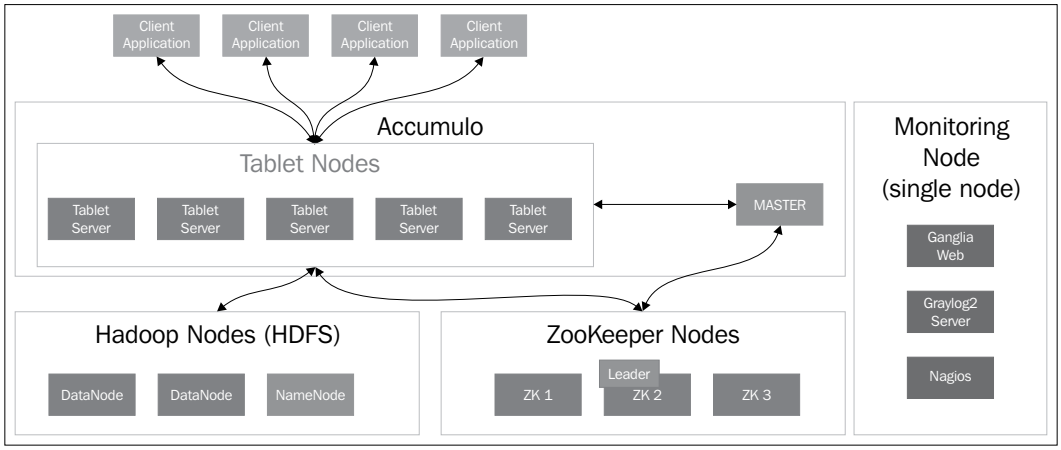
- Monitoring
- Elasticity
- Failover
- Resource management

Monitoring

The main reason for monitoring clusters in general is to find problems and fix them. For a simple setup, there is no need to set up Ganglia, but if it is already set up, there is no reason not to use it. The same rule applies to Nagios. The Graylog2 server is a tool that I would always use no matter how small or large the cluster setup is.

In the case of setting up Accumulo on the developer's machine, the Graylog2 server will save a lot of time in correlating all of the logs from Accumulo, ZooKeeper, and Hadoop, and present them through an easy-to-use web interface.

Before continuing, let's establish a baseline for the rest of this chapter and imagine we have the setup, as depicted in the following figure:



But what is Ganglia and how can it help to find problems in the cluster? Ganglia is a very powerful application, and it runs on Windows, Linux, Mac OS, and many others. Ganglia also provides almost real-time monitoring for very large networks that are usually too large to monitor with traditional monitoring systems. Ganglia scales from a small cluster to an extremely large cluster, such as tens of thousands of machines. If you haven't used Ganglia before and are running a large cluster, you should at least consider using it.

Setting up Ganglia

Ganglia is composed of three packages:

- **Ganglia monitor core** (Version 3.6) contains the gmond, gmetad, PHP web frontend, gmetric, gstat, and libganglia components
- **Ganglia web 2.0** (Version 3.5.1) contains the Ganglia web component
- **gexec execution environment** (Version 0.2.1/0.3.4) contains the gexec, gexecd, authd, and libe components

For the latest version of Ganglia packages, visit the following page:

`http://ganglia.info/?page_id=66`

For all of the Accumulo, Hadoop, and ZooKeeper nodes, install the **Ganglia monitor deamon** (**gmond**), by using the following command:

```
sudo apt-get install ganglia-monitor
```

Next, you need to set up the **Ganglia meta deamon** (**gmetad**) for all of the Accumulo, Hadoop, and ZooKeeper nodes. Ganglia meta deamon collects metric data from another Ganglia meta deamon and Ganglia monitor deamon, and then stores them to a disk:

```
sudo apt-get install gmetad
```

Next, you need to set up Ganglia web; but before that, Ganglia web requires the following to be installed:

- Apache Web Server
- PHP 5.2 and later
- PHP JSON

Run the following command to install the required software (you need to enable the PHP JSON module):

```
sudo apt-get install apache2 php5 php5-json
```

To set up Ganglia web, follow the guidelines in the following link:

`http://sourceforge.net/apps/trac/ganglia/wiki/ganglia-web-2`

Configuring Ganglia

For every Accumulo, Hadoop, and ZooKeeper node, you might need to change the following two files:

- `/etc/ganglia/gmetad.conf` – Add the following line to this file and list the data source:

  ```
  data_source "Accumulo Cluster" host1 host2 … hostx
  ```

- `/etc/ganglia/gmond.conf` – For every host you want to send UDP packages to, you need to add the following section:

  ```
  udp_send_channel {
    host = <the hostname>
    port = 8659
    ttl = 1
  }
  ```

Setting up the Graylog2 server

When setting up an instance of the Graylog2 server, the easiest way is to use scripts. There are many good scripts, but I usually use scripts located in this Git repository at `https://github.com/mrlesmithjr/graylog2`. Please follow the instructions in the `README` file.

Logging using Graylog2

You can send a standard syslog via TCP/UDP, and GELF via UDP, TCP, and HTTP to the Graylog2 server. There are shortcomings of a classic syslog, so the recommended way is to use **Graylog Extended Log Format** (**GELF**), because Accumulo, Hadoop, and ZooKeeper are written in Java and they use **log4j**. The only thing we need to do is perform the following steps:

1. Download gelf4j from `https://github.com/pstehlik/gelf4j`.
2. Build gelf4j.
3. Place the `gelf4j.jar` file in your classpath.
4. Change the following files:
    - `/usr/local/accumulo/conf/log4j.properties`
    - `/usr/local/hadoop/conf/log4j.properties`
    - `/usr/local/zookeeper/conf/log4j.properties`

    And change the following lines:

```
# GELF appender
log4j.appender.GELF=com.pstehlik.groovy.gelf4j.appender.
Gelf4JAppender
log4j.appender.GELF.graylogServerHost=<your graylog server>
log4j.appender.GELF.host=<this host machine name>
log4j.appender.GELF.facility=GELF
```

Setting up Nagios

Nagios comes as an easy-to-install package; executing the following command will get Nagios up and running:

```
sudo apt-get install -y nagios3
```

Hadoop

In our demo cluster, we have one Hadoop NameNode and two Hadoop DataNodes. While examining the status of Hadoop, we use the web interface on the NameNode.

NameNode web interface

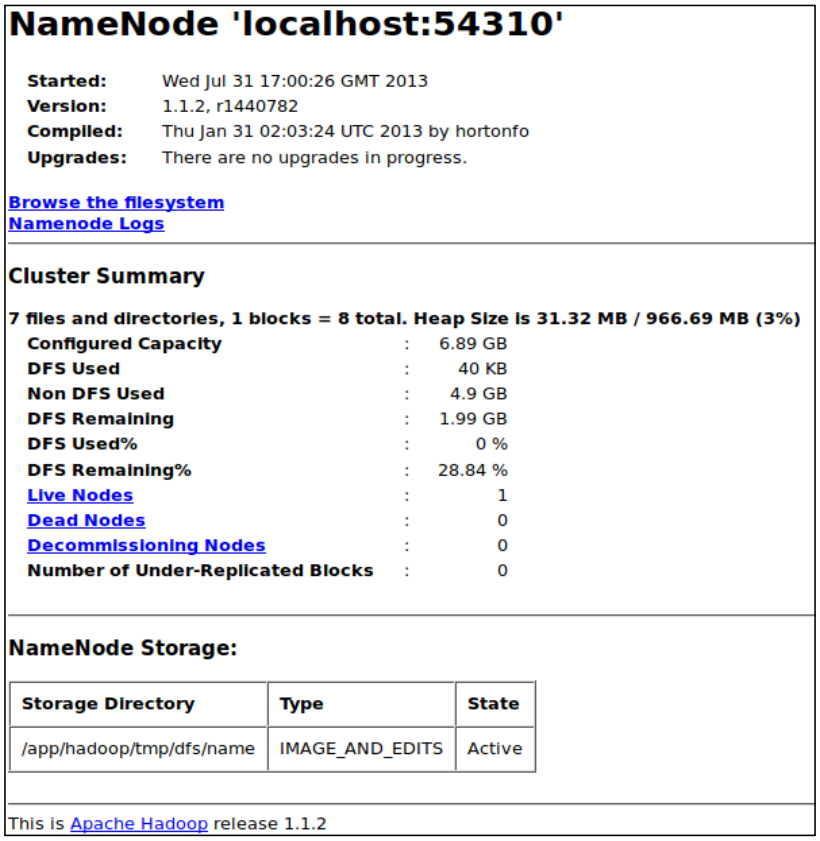
For the NameNode, the Web shows crucial information about cluster summary. In the cluster summary, you will find information about live nodes, dead nodes, decommissioning nodes, and total/remaining capacity. There is also a link to browse the filesystem and look at logfiles.

It is very common to see the number of dead nodes greater than zero. When that happens, you need to react. In the next version of Hadoop (2.1.0, currently in beta), there is the Resource Management REST API that gives all of the information that would allow us to monitor clusters with Nagios. That means, instead of monitoring the resources through Graylog2 with triggers, we can use Nagios. It can monitor a response from the URL. In the next version of Hadoop (2.1.0), the Resource Manager API exposes the following:

API	Description	URI
Cluster information API	This resource provides overall information about the cluster.	`http://<rm http address:port>/ws/v1/cluster/info`
Cluster metrics API	This resource provides overall metrics about the cluster.	`http://<rm http address:port>/ws/v1/cluster/metrics`
Cluster scheduler API	This resource provides information about the current scheduler in a cluster.	`http://<rm http address:port>/ws/v1/cluster/scheduler`
Cluster applications API	This resource provides information about the collection of applications.	`http://<rm http address:port>/ws/v1/cluster/apps`
Cluster application API	This resource provides information about an application.	`http://<rm http address:port>/ws/v1/cluster/apps/{appid}`
Cluster application attempts API	This resource provides a collection of applications that represent an application attempt.	`http://<rm http address:port>/ws/v1/cluster/apps/{appid}/appattempts`

API	Description	URI
Cluster nodes API	This resource provides a collection of nodes.	`http://<rm http address:port>/ws/v1/cluster/nodes`
Cluster node API	This resource provides information about nodes.	`http://<rm http address:port>/ws/v1/cluster/nodes/{nodeid}`

As shown in the following screenshot the NameNode web is very simple and only shows important information like the cluster summary:



Finding the logfiles

Logfiles in Hadoop can be found through the NameNode web interface. Browsing through the NameNode logs, you will find the following files and folders:

- `hadoop-hadoopuser-datanode-<machinename>.log`
- `hadoop-hadoopuser-jobtracker-<machinename>.log`

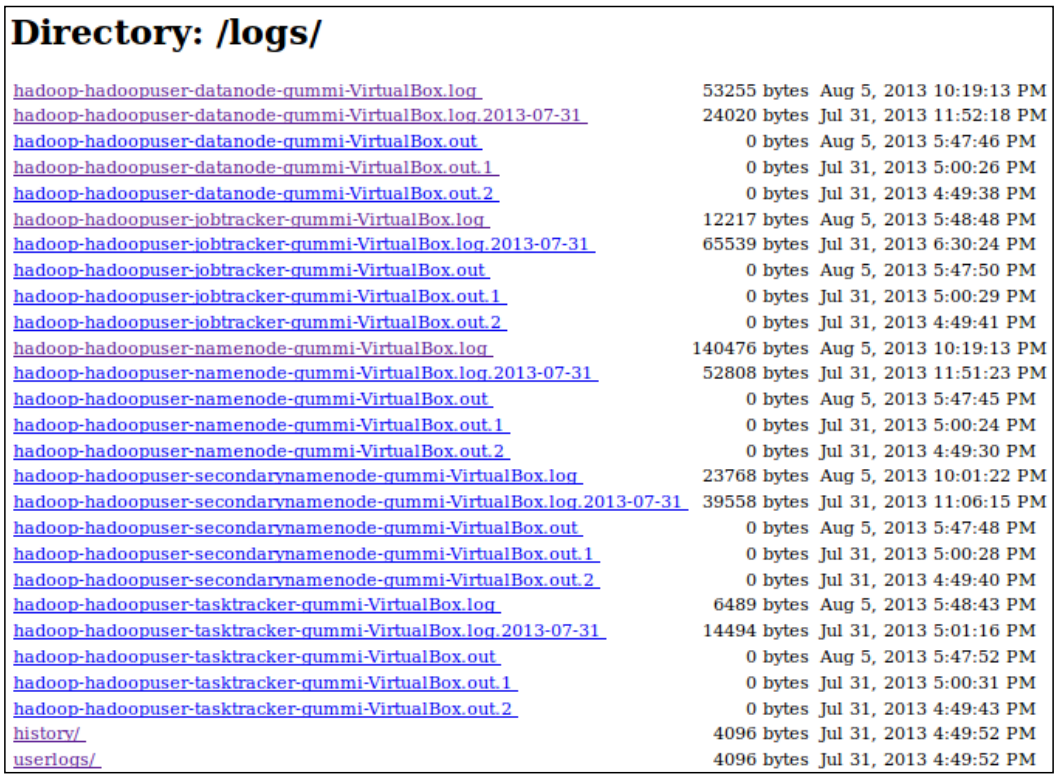
- `hadoop-hadoopuser-namenode-<machinename>.log`
- `hadoop-hadoopuser-secondarynamenode-.<machinename>.log`
- `hadoop-hadoopuser-tasktracker-<machinename>.log`
- `history/`
- `userlogs/`

To enable Graylog2 as the logfile monitoring tool, you need to change the following files:

- `/usr/local/hadoop/conf/log4j.properties`
- `/usr/local/accumulo/conf/log4j.properties`
- `/usr/local/zookeeper/conf/log4j.properties`

Add the Graylog2 appender and point to the Graylog2 machine, as explained in the *Logging using Graylog2* section in this chapter.

As shown in the following screenshot the `logs` directory gets cluttered very quickly, even when only running single node. The naming rule helps finding the correct logfile.



Directory: /logs/

hadoop-hadoopuser-datanode-gummi-VirtualBox.log	53255 bytes	Aug 5, 2013 10:19:13 PM
hadoop-hadoopuser-datanode-gummi-VirtualBox.log.2013-07-31	24020 bytes	Jul 31, 2013 11:52:18 PM
hadoop-hadoopuser-datanode-gummi-VirtualBox.out	0 bytes	Aug 5, 2013 5:47:46 PM
hadoop-hadoopuser-datanode-gummi-VirtualBox.out.1	0 bytes	Jul 31, 2013 5:00:26 PM
hadoop-hadoopuser-datanode-gummi-VirtualBox.out.2	0 bytes	Jul 31, 2013 4:49:38 PM
hadoop-hadoopuser-jobtracker-gummi-VirtualBox.log	12217 bytes	Aug 5, 2013 5:48:48 PM
hadoop-hadoopuser-jobtracker-gummi-VirtualBox.log.2013-07-31	65539 bytes	Jul 31, 2013 6:30:24 PM
hadoop-hadoopuser-jobtracker-gummi-VirtualBox.out	0 bytes	Aug 5, 2013 5:47:50 PM
hadoop-hadoopuser-jobtracker-gummi-VirtualBox.out.1	0 bytes	Jul 31, 2013 5:00:29 PM
hadoop-hadoopuser-jobtracker-gummi-VirtualBox.out.2	0 bytes	Jul 31, 2013 4:49:41 PM
hadoop-hadoopuser-namenode-gummi-VirtualBox.log	140476 bytes	Aug 5, 2013 10:19:13 PM
hadoop-hadoopuser-namenode-gummi-VirtualBox.log.2013-07-31	52808 bytes	Jul 31, 2013 11:51:23 PM
hadoop-hadoopuser-namenode-gummi-VirtualBox.out	0 bytes	Aug 5, 2013 5:47:45 PM
hadoop-hadoopuser-namenode-gummi-VirtualBox.out.1	0 bytes	Jul 31, 2013 5:00:24 PM
hadoop-hadoopuser-namenode-gummi-VirtualBox.out.2	0 bytes	Jul 31, 2013 4:49:30 PM
hadoop-hadoopuser-secondarynamenode-gummi-VirtualBox.log	23768 bytes	Aug 5, 2013 10:01:22 PM
hadoop-hadoopuser-secondarynamenode-gummi-VirtualBox.log.2013-07-31	39558 bytes	Jul 31, 2013 11:06:15 PM
hadoop-hadoopuser-secondarynamenode-gummi-VirtualBox.out	0 bytes	Aug 5, 2013 5:47:48 PM
hadoop-hadoopuser-secondarynamenode-gummi-VirtualBox.out.1	0 bytes	Jul 31, 2013 5:00:28 PM
hadoop-hadoopuser-secondarynamenode-gummi-VirtualBox.out.2	0 bytes	Jul 31, 2013 4:49:40 PM
hadoop-hadoopuser-tasktracker-gummi-VirtualBox.log	6489 bytes	Aug 5, 2013 5:48:43 PM
hadoop-hadoopuser-tasktracker-gummi-VirtualBox.log.2013-07-31	14494 bytes	Jul 31, 2013 5:01:16 PM
hadoop-hadoopuser-tasktracker-gummi-VirtualBox.out	0 bytes	Aug 5, 2013 5:47:52 PM
hadoop-hadoopuser-tasktracker-gummi-VirtualBox.out.1	0 bytes	Jul 31, 2013 5:00:31 PM
hadoop-hadoopuser-tasktracker-gummi-VirtualBox.out.2	0 bytes	Jul 31, 2013 4:49:43 PM
history/	4096 bytes	Jul 31, 2013 4:49:52 PM
userlogs/	4096 bytes	Jul 31, 2013 4:49:52 PM

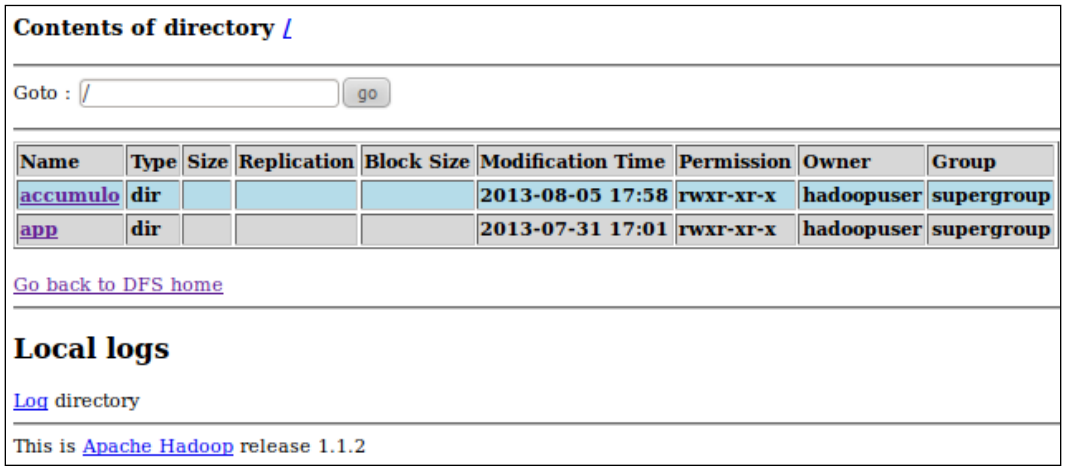
How does Accumulo store files in Hadoop?

Browsing through the filesystem will give you a good overview on how Accumulo stores files in Hadoop HDFS:

- `/accumulo`
  - `instance_id`
  - `recovery`
  - `tables`
  - `versions`
  - `walogArchive`
- `/app`
- `hadoop` – This is the default folder we created in *Chapter 1, Building an Accumulo Cluster from Scratch*

We will go more in depth on the Accumulo file structure in later chapters.

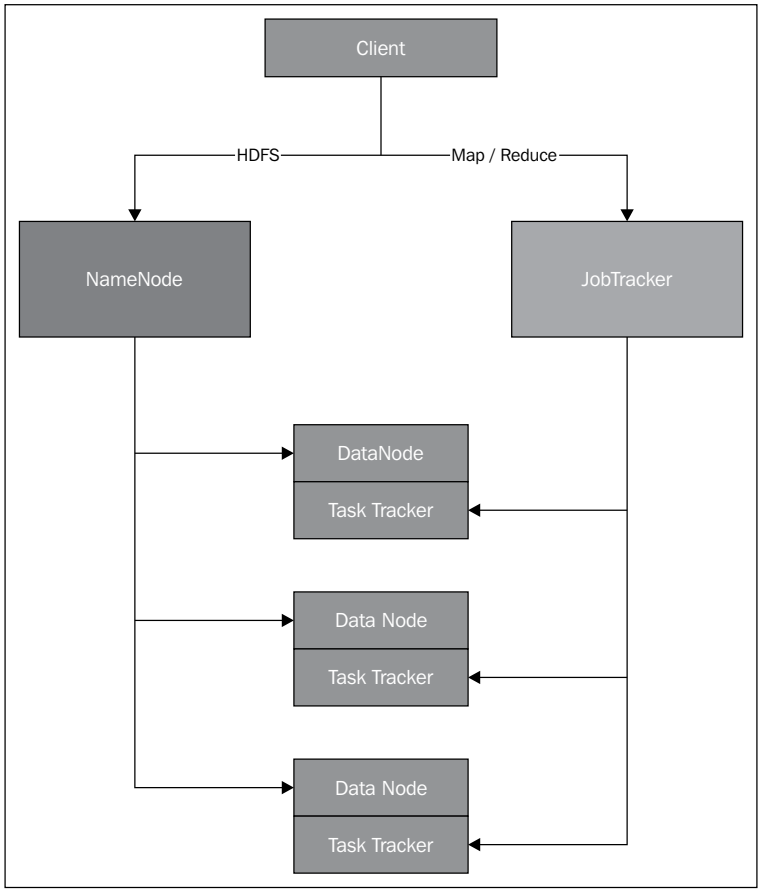
Hadoop allows browsing the data directories through a simple web interface as shown in the following screenshot. It is good practice to get familiar with Hadoop directory structure through this web interface.



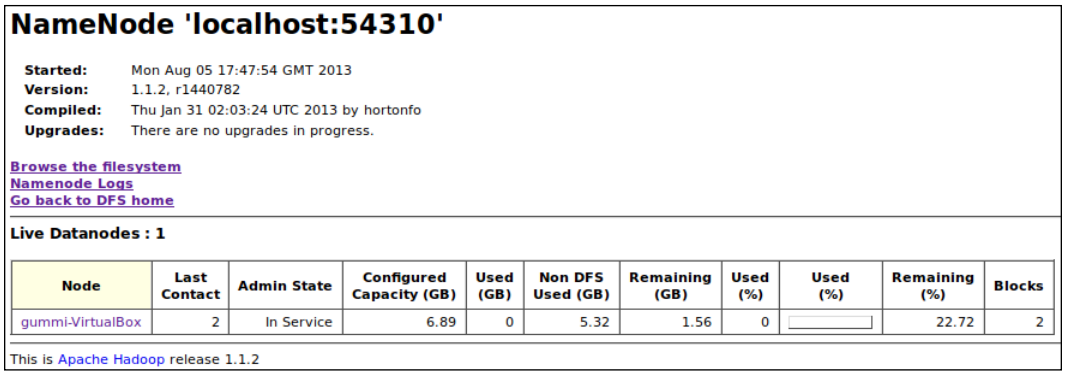
Live, dead, and decommissioning nodes

In Hadoop, you have NameNodes and DataNodes. Hadoop has a single point of failure if the NameNode goes down. For redundancy, set up a secondary NameNode. At the same time, Hadoop is designed to be fault-tolerant when DataNode goes down. Hadoop supports the decommissioning of nodes, that is, to retire an existing DataNode or even a set of existing DataNodes.

As shown in the following figure the relationship between HDFS and Map/Reduce in Hadoop isn't complex:



One of the greatest views in the NameNode web interface is the visibility of live, dead, and decommissioning nodes. As Hadoop is designed to run on top of cheap hardware, and hardware failure is a norm rather than an exception, you need to watch the nodes carefully and know what is happening by using Graylog2.



NameNode 'localhost:54310'

Started: Mon Aug 05 17:47:54 GMT 2013
Version: 1.1.2, r1440782
Compiled: Thu Jan 31 02:03:24 UTC 2013 by hortonfo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

Live Datanodes : 1

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
gummi-VirtualBox	2	In Service	6.89	0	5.32	1.56	0		22.72	2

This is Apache Hadoop release 1.1.2

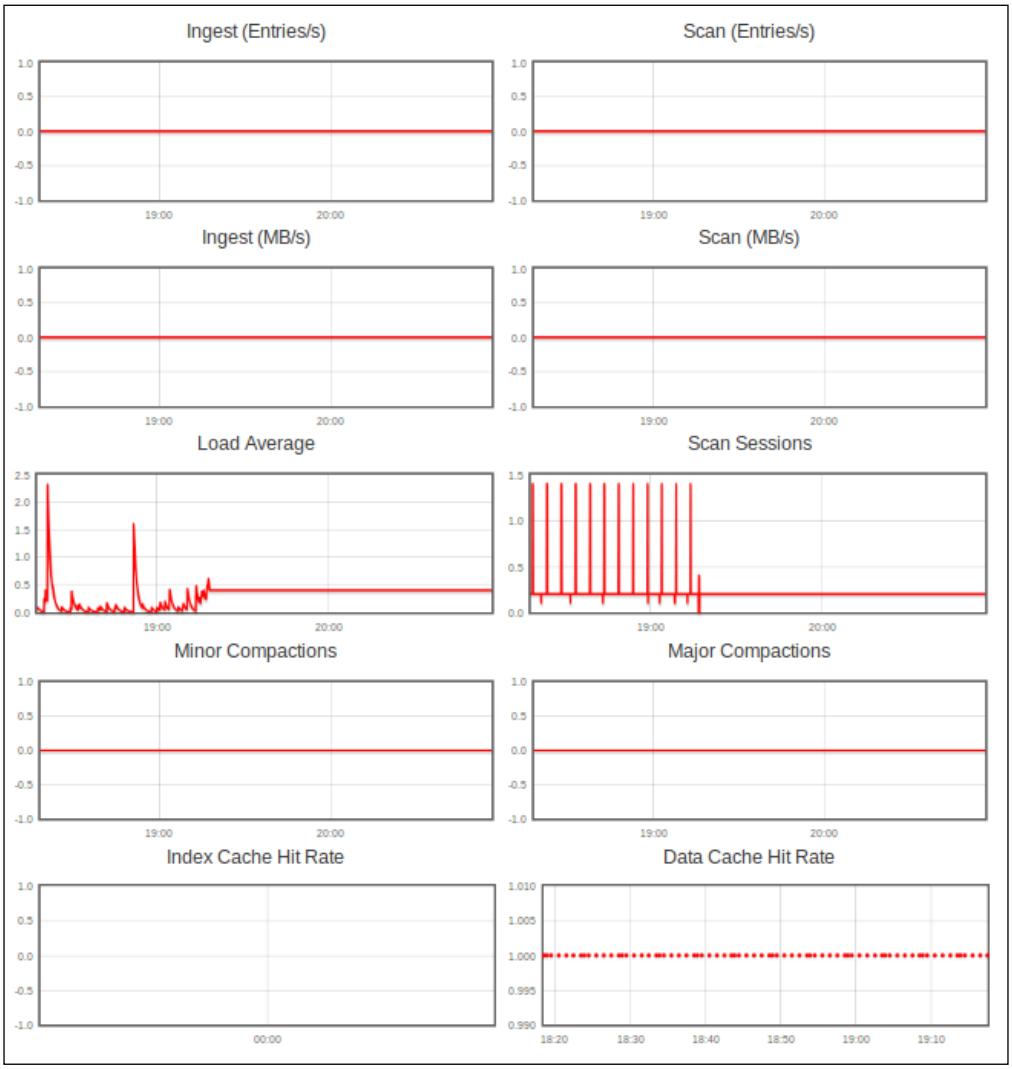
Accumulo

Before we do anything more, let's examine what kind of information we can get from the standard interface for Accumulo `http://localhost:50095`. Refer to *Chapter 1*, *Building an Accumulo Cluster from Scratch*, for more information on Accumulo. Also, it gives you information about the following:

- **Accumulo Master**: When a failure occurs on TabletServer, Accumulo Master is responsible for a response. Failure is handled by balancing the load with another TabletServer.
- **TabletServer**: This manages many tablets (partitions of tables). A tablet is a unit of work for the TabletServer and has a row range within a table.
- **NameNode**: This is the key information about the Hadoop NameNode.
- **JobTracker**: This shows running jobs, map tasks, reduce tasks, trackers, and blacklisted if any for Accumulo.
- **ZooKeeper**: This shows how many ZooKeeper nodes are available for Accumulo and how many clients are currently connected.
- **Graphs/Performance numbers**
    - Ingest (Entries/s)
    - Scan (Entries/s)
    - Ingest (MB/s)
    - Scan (MB/s)
    - Load average
    - Scan sessions
    - Minor compactions

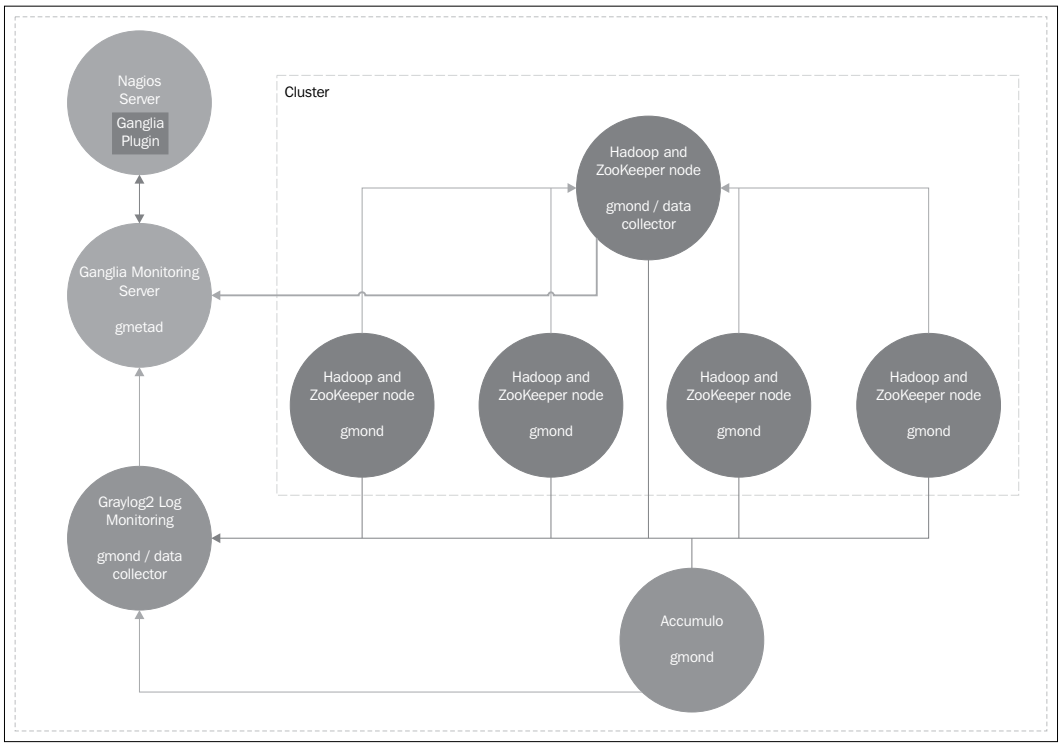
- Major compactions
- Index cache hit rate
- Data cache hit rate

Accumulo performance numbers will be covered in more detail in *Chapter 4, Optimizing Accumulo Performance*.



Monitoring a system's overview

The following figure shows an example where a cluster is monitored with Nagios, Ganglia, and Graylog2 to monitor the entire cluster:



We have one or two gathering machine(s) to create a notion of two clusters, one for Hadoop (HDFS) and ZooKeeper, and another for Accumulo. To hook Nagios and Ganglia together, we need to install a plugin into Nagios.

Elasticity

Hadoop allows you to execute every task in parallel, and the only concern you should have is how many machines are available, and what is the optimal number of machines to use when performing a Map/Reduce job. Instead, the problem is more a question of how you can import and export data from the Hadoop cluster. Accumulo solves this problem by giving applications BigTable access to the Hadoop filesystem.

Accumulo operates over the Hadoop Distributed File System (HDFS). Accumulo supports efficient storage and retrieval of structured data, including queries for ranges, and provides support for using Accumulo tables as the input and output for Map/Reduce jobs.

Failover

By design, Hadoop's single point of failure is when the NameNode goes down, and there isn't a secondary NameNode running. This limitation requires Hadoop to be configured for NameNode failover. For more information about NameNode failover, visit `http://wiki.apache.org/hadoop/NameNodeFailover`.

When nodes in the ZooKeeper cluster are started, the first task the nodes do is to find the leader. After the leader has been chosen, the rest of the ZooKeeper nodes will follow that leader. The concept of "follow the leader" means that there is always a leader machine and (in a multi-node environment), some followers. The follower can be behind the leaders, but that will not affect the clients because they will always connect to the same ZooKeeper machine. A ZooKeeper cluster is self-healing, and after the ZooKeeper server restarts, it will rejoin the cluster. For self-healing to work, you need to have a supervisory process to monitor each server's processes.

Resource management

In the upcoming version of Hadoop (2.1.0), the resource manager REST APIs have been added and will give us information about the cluster, status of the cluster, metrics of the cluster, scheduler information, information about nodes in the cluster, and information about applications in the cluster. In the current version of Hadoop, this luxury doesn't exist.

Summary

Monitoring is the key when it comes to elasticity, failover, and resource management. To be able to see what the system is doing in real time is what is often needed to find problems, and to avoid problems if possible. Tools such as Ganglia, Nagios, and the Graylog2 server are helping many companies running large clusters with a minimum downtime.

3
Integrating Accumulo into Various Cloud Platforms

In this chapter, we will learn how to integrate Accumulo into various cloud platforms, both as a single-node and as a pseudo-distributed mode, and then expand it to a multi-node. This is similar to *Chapter 1*, *Building an Accumulo Cluster from Scratch* (refer to *Setting up Hadoop*, *Setting up ZooKeeper*, and *Setting up and configuring Accumulo*), where the aim was to build an **Accumulo** cluster from scratch, but with a focus on integration with various cloud platforms.

The following example will show you how to create an Accumulo cluster on various cloud platforms. The steps needed to complete the task of setting up the cluster are similar for those cloud platforms. The difference is in the tools and scripts used to accomplish the task of creating the cluster.

These are the topics that will be covered in this chapter:

- Amazon EC2
- Google Cloud Platform
- Rackspace
- Windows Azure

Hadoop is supported by many cloud vendors as the popularity of Map/Reduce has grown over the past few years. Accumulo is another story; even though popularity is growing, the support of cloud vendors hasn't caught up.

Amazon EC2

Amazon has great support for **Accumulo**, **Hadoop**, and **ZooKeeper**. For Hadoop and ZooKeeper, there is a set of libraries called **Apache Whirr.** Apache Whirr supports Amazon EC2, Rackspace, and many more cloud providers. Apache Whirr uses low-level API libraries. For Accumulo, you have two options: one is to use the Amazon EMR command-line interface, and the other is to create a new virtual machine and setup as explained in *Chapter 1*, *Building an Accumulo Cluster from Scratch*.

Prerequisites for Amazon EC2

Prerequisites needed to complete the setup phase for Amazon EC2 are as follows:

- Cygwin is required
- Windows users need to download and install **PuTTY** from `http://www.putty.org/or use Cygwin SSH`
- A valid user is needed to access **Amazon AWS Console**
- Install the **Amazon EMR command-line interface** by following the steps at this location, `http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-cli-install.html`

Creating Amazon EC2 Hadoop and ZooKeeper cluster

The following steps are required to create Amazon EC2 Hadoop and the ZooKeeper cluster:

1. Log in to `https://console.aws.amazon.com`.
2. The management console for Amazon Services has a nice graphical overview of all the actions that you can do. In our case, we use the **Amazon AWS Console** to verify what we have done while setting up the cluster.
3. From the drop-down menu under your name at the top-right corner, select **Security Credentials**.
4. Under **Access Keys**, you need to create a new root key and download the file containing `AWSAccessKeyId` and `AWSSecretKey`.
5. Normally, you would create an AWS **Identity and Access Management** (**IAM**) user with limited permissions, and give only that user the access to the cluster. But in this case, we are creating a demo cluster and will be destroying it after use.

6. Create a new key by running the following command:
   - For Linux and Windows Cygwin:

     ```
     ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa_whirr
     ```

   The `rsa` key is used later when configuring Whirr. It is not required to copy the key to the `~/.ssh/authorized_keys` folder because the `rsa` key is going to be used from the current location.

7. Download Whirr and set it up using the following commands:

   ```
   cd /usr/local
   sudo wget http://apache.claz.org/whirr/stable/whirr-0.8.2.tar.gz
   sudo tar xzf whirr-0.8.2.tar.gz
   sudo mv whirr-0.8.2 whirr
   sudo chown -R hadoopuser:hadoopgroup whirr
   ```

   Download Whirr in the `/usr/local` folder, unpack it, and rename it to `whirr`. For Cygwin, don't run the last command in the script.

8. Set up the credentials for Amazon EC2:
   - For Linux and Cygwin:

     ```
     sudo cp /usr/local/whirr/conf/credentials.sample
        /usr/local/whirr/credentials
     sudo nano /usr/local/whirr/conf/credentials
     ```

   - Skip the `sudo` command in Cygwin. Elevated privileges in Windows are usually acquired by right-clicking on the icon and choosing **Run as administrator**.
   - Edit the `/usr/local/whirr/const/credentials` file and change the following lines:

     ```
     PROVIDER=aws-ec2
     IDENTITY=<The value from the variable AWSAccessKeyId>
     CREDENTIAL= <The value from the variable AWSSecretKey>
     ```

   - By default, Whirr will look for the `credentials` file in the home directory; if it's not found there, it will look in `/usr/local/whirr/conf`. I prefer to use the `/usr/local/whirr/conf` directory to keep everything at the same place.

9. The first step in simplifying the creation of the cluster is to create a configuration file, which will be named `cluster.properties` for this example.
    - For Linux:

      ```
      sudo nano /usr/local/whirr/conf/cluster.properties
      ```

    - For Cygwin:

      ```
      nano /usr/local/whirr/conf/cluster.properties
      ```

    Add the following lines:

    ```
    whirr.cluster-name=demo-cluster
    whirr.instance-templates=1 zookeeper,1 hadoop-
       jobtracker+hadoop-namenode,1 hadoop-datanode+Hadoop
       -tasktracker
    whirr.provider=aws-ec2
    whirr.private-key-file=${sys:user.home}/.ssh/id_rsa_whirr
    whirr.public-key-file=
       ${sys:user.home}/.ssh/id_rsa_whirr.pub
    ```

    This file describes a single cluster with one ZooKeeper node, one Hadoop node running JobTracker and NameNode, and one Hadoop node running DataNode and JobTracker.

10. Create our cluster as described in the `cluster.properties` file:
    - For Linux:

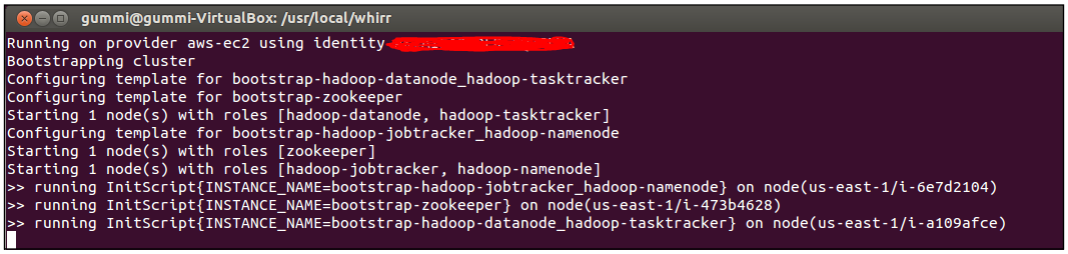
      ```
      su - hadoopuser
      ```

    - For Windows Cygwin:

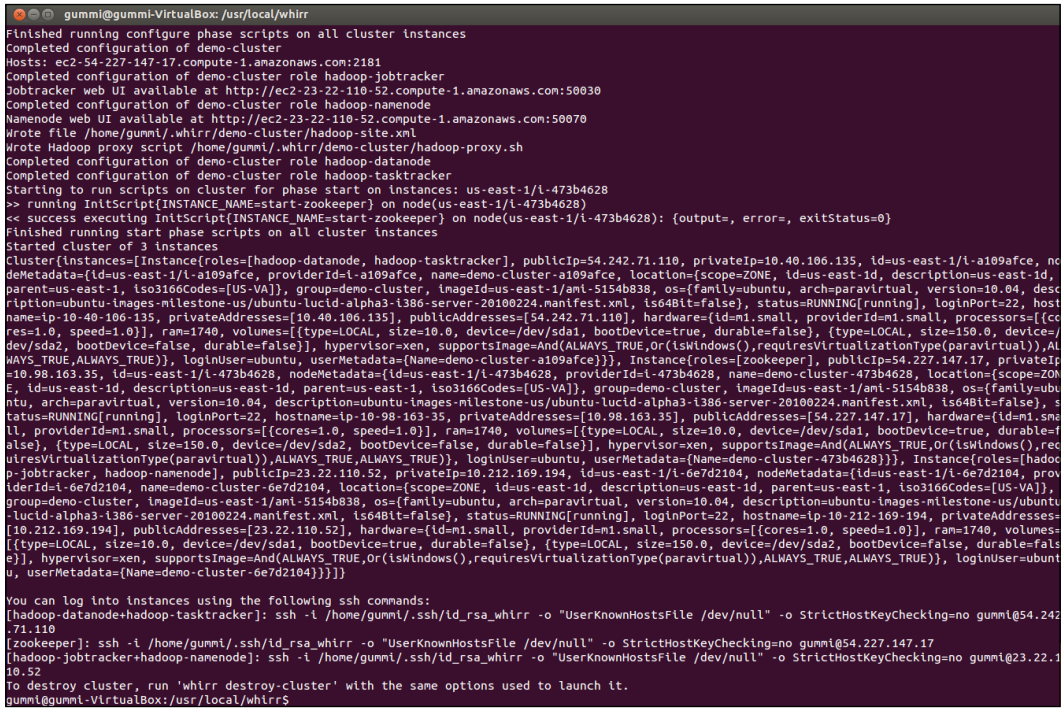
      ```
      cd /usr/local/whirr
      bin/whirr launch-cluster --config conf/cluster.properties
      ```

    If you get the error message `java.io.FileNotFoundException: whirr.log (Permission denied)`, then the current user has not got permission to access the `whirr.log` file.

After a few seconds, you will see that the script will start to print out the status message and information about what is going to be done, as shown in the following screenshot:

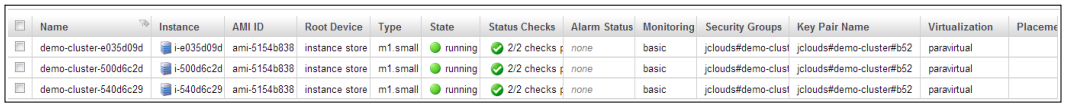


The result from creating a cluster using Whirr is very detailed and important for troubleshooting and monitoring purposes, as shown in the following screenshot:



The output from running the script gives very valuable information about the cluster created. Every instance has a role and an external and internal IP address. The ID of every node is in the form <region>/<unique id>.

11. After creating the cluster, please visit `https://console.aws.amazon.com/ec2/home?region=us-east-1#s=Instancesto` to see your new cluster. If the cluster was created in another region, change it to the correct region at the top.



Name	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status	Monitoring	Security Groups	Key Pair Name	Virtualization	Placeme
demo-cluster-e035d09d	i-e035d09d	ami-5154b838	instance store	m1.small	running	2/2 checks p	none	basic	jclouds#demo-clust	jclouds#demo-cluster#b52	paravirtual	
demo-cluster-500d6c2d	i-500d6c2d	ami-5154b838	instance store	m1.small	running	2/2 checks p	none	basic	jclouds#demo-clust	jclouds#demo-cluster#b52	paravirtual	
demo-cluster-540d6c29	i-540d6c29	ami-5154b838	instance store	m1.small	running	2/2 checks p	none	basic	jclouds#demo-clust	jclouds#demo-cluster#b52	paravirtual	

12. Destroy our cluster as described in the `cluster.properties` file, by running the following command for Linux and Windows Cygwin:

```
cd /usr/local/whirr
bin/whirr destroy-cluster --config conf/cluster.properties
```

13. The directory `~/.whirr/demo-cluster` has been created as a direct result of the previous step, and contains information about the cluster just created and three files:
    - `hadoop-proxy.sh`: Run this script to create a proxy tunnel to be able to connect to the cluster using the SSH tunnel. Use this example to create a proxy auto-config (PAC) file: `https://svn.apache.org/repos/asf/whirr/trunk/resources/hadoop-ec2-proxy.pac`.
    - `hadoop-site.xml`: It contains information about the Hadoop cluster.
    - `instances`: It contains information about each node instance (location, instance, role(s), external IP address, and internal IP address).
14. All nodes in the preceding example were created in the same security group that allows them to talk to each other.

Setting up Accumulo

The easiest way to set up Accumulo on Amazon is to use the Amazon CLI (command-line interface). There is a single ZooKeeper node up and running, that should be used while setting up Accumulo.

1. Browse to the Amazon EC2 console `https://console.aws.amazon.com/s3/home?region=us-east-1#`, and create a new bucket with a unique name. For this example, the name `demo-accumulo` will be used.
2. To create an instance of Accumulo, we use the following commands in Amazon CLI:

For Linux and Windows:

```
elastic-mapreduce --create --alive --name "Accumulo"
   --bootstrap-action \

s3://elasticmapreduce/samples/accumulo/accumulo-install.sh \

--args "<zookeeper ip address>, Demo-Database, DBPassword"
   --bootstrap-name "install Accumulo" \

--enable-debugging -log-url s3://demo-accumulo/Accumulo-logs/\

--instance-type m1.large --instance-count 4 --key- pair
   <Key Pair Name>
```

Locate the key pair name at `https://console.aws.amazon.com/ec2/home?region=us-east-1#s=KeyPairs`.

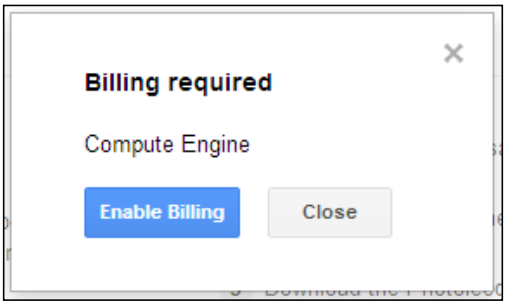
Google Cloud Platform

Accumulo design is based on Google's **BigTable** design published in 2006, therefore, you will see lot of similarities between Google and Accumulo with respect to performing a search. Google doesn't have the same support for Hadoop as Amazon but you can easily perform the same tasks. One of the trademarks Google has is the simple user interface for the **Google Cloud Console**.

Prerequisites for Google Cloud Platform

The prerequisites for Google Cloud Platform are:

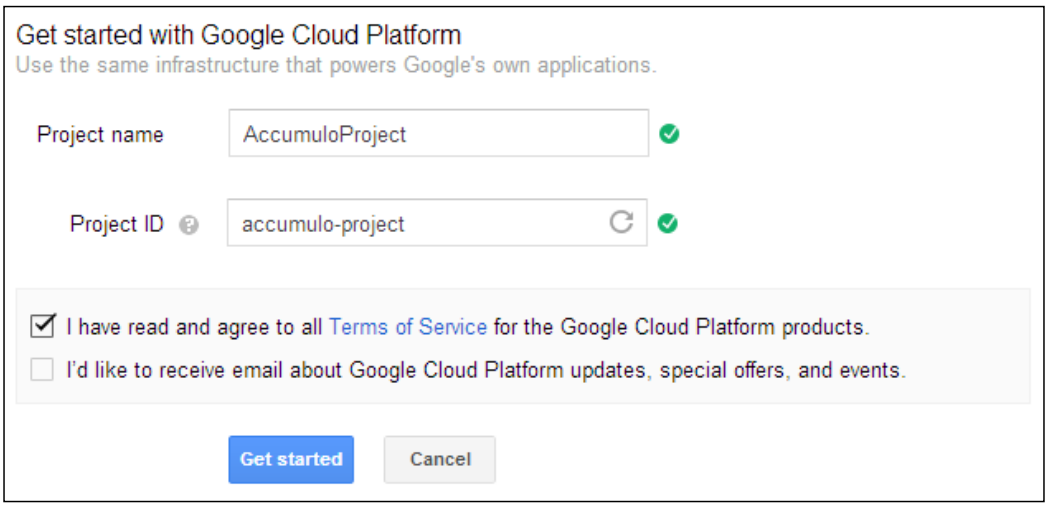
- A valid user to access the Google Cloud Console. Remember billing is required to continue, as shown in the following screenshot:



- Downloading and installing Python 1.7.x

Creating the project

Everything in Google Cloud Platform revolves around the project. The first task that we need to do is to create a new project. We are going to name the project as `AccumuloProject` with the **Project ID** `accumulo-project`, as shown in the following screenshot. Of course, a more descriptive name would be used in practice:



Installing the Google gcutil tool

Now, you need to install the Google **gcutil** tool from `https://developers.google.com/compute/docs/gcutil/, in order` to continue. The gcutil tool has a lot of useful commands, but we are only going to focus on the commands that we need to complete our tasks. In the following examples, gcutil is installed in the `/usr/local/gsutil` directory.

Configuring credentials

After setting up the gcutil tool, you need to run the following commands:

For Linux:

```
/usr/local/gcutil/gcutil auth --project=accumulo-project
```

For Windows (Cygwin):

```
python /usr/local/gcutil/gcutil.py auth --project=accumulo-project
```

On running this command, you will be prompted to open the website to get the verification code that you need to enter (or copy from the webpage).

Configuring the project

To simplify the usage of gcutil, we are using the flag `--cache_flag_values`. This will cause the file `~/.gcutil.flags` to be created, and the default project ID will be stored in that file.

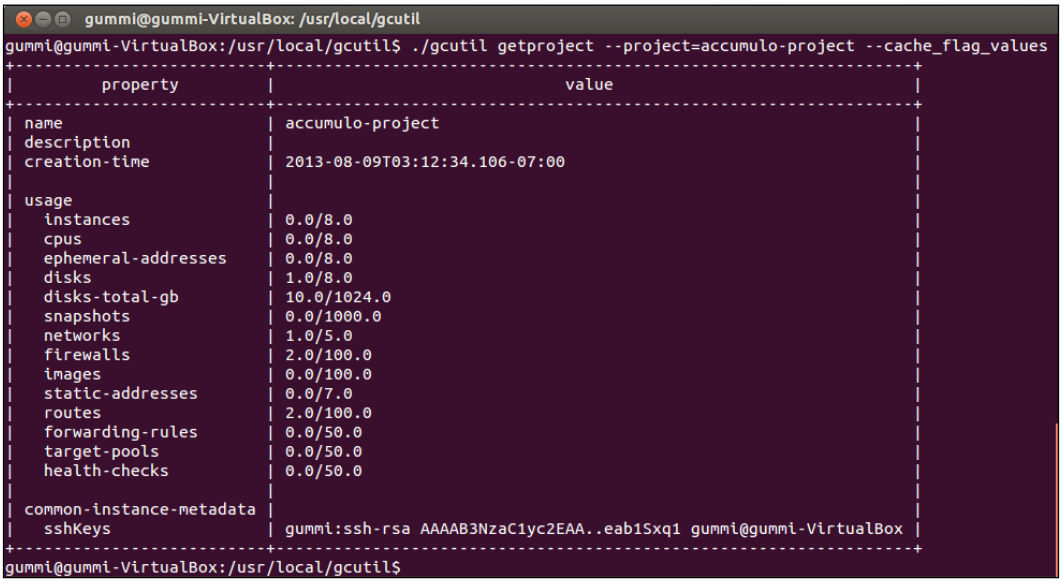
For Linux, use the following command:

```
/usr/local/gcutil/gcutil getproject --project=accumulo-project
   --cache_flag_values
```

For Windows Cygwin, use the following command:

```
python /usr/local/gcutil/gcutil.py getproject
   --project=accumulo-project --cache_flag_values
```

After running this command, you should get a report like the following:



Creating the firewall rules

We need to create firewall rules that permit incoming HTTP traffic on ports 50030, 50060, and 50070. The easiest way to accomplish that is through the Google Cloud Console.

Creating the cluster

Google Cloud only supports Debian and CentOS Linux. In this example, CentOS-6-v20130813 is going to be used, but this changes on a regular basis, and you need to see what images are available before starting.

The boot disk is unspecified. I can create a new persistent boot disk and use it (preferred), or use a scratch disk (not recommended). Answer the following question with a `Y` (yes) when asked during the setup process:

```
Do you want to use a persistent boot disk? [y/n]
```

Creating the cluster involves four actions:

- Create and set up a Hadoop NameNode. A new instance is created, and then Hadoop is set up and started as a master.
- Create and set up a Hadoop DataNode. A new instance is created, and then Hadoop is set up and started as a slave.
- Create and set up a ZooKeeper node. A new instance is created, and then ZooKeeper is set up and started.
- Create and set up an Accumulo node. A new instance is created, and then Accumulo is set up and started.

Hadoop

Create two nodes: NameNode and DataNode.

- Create the Hadoop NameNode:
    - For Linux:

```
/usr/local/gcutil/gcutil addinstance hadoop-namenode
   --machine_type=n1-standard-1 --image=centos-6-v20130731
   --zone=europe-west1-b
```

    - For Windows (Cygwin):

```
python /usr/local/gcutil/gcutil.py addinstance hadoop-
namenode
   --machine_type=n1-standard-1 --image=centos-6-v20130731
   --zone=europe-west1-b
```

- Connect to the newly created Hadoop NameNode:
    - For Linux:

```
/usr/local/gcutil/gcutil --service_version="v1beta15"
   --project="accumulo-project" ssh  --zone="europe-west1-b"
   "hadoop-namenode"
```

    - For Windows:

```
python /usr/local/gcutil/gcutil --service_version="v1beta15"
   --project="accumulo-project" ssh  --zone="europe-west1-b"
   "hadoop-namenode"
```

- Follow the guidelines given in the *Setting up Hadoop* section in *Chapter 1, Building an Accumulo Cluster from Scratch*, to set up Hadoop (master).
- Create the Hadoop DataNode:
    - For Linux:

```
/usr/local/gcutil/gcutil addinstance hadoop-datanode
   --machine_type=n1-standard-1 --image=centos-6-v20130731
   --zone=europe-west1-b
```

    - For Windows (Cygwin):

```
python /usr/local/gcutil/gcutil.py addinstance hadoop-
datanode   --machine_type=n1-standard-1
--image=centos-6-v20130731--zone=europe-west1-b
```

- Connect to the newly created Hadoop DataNode:
    - For Linux:

```
/usr/local/gcutil/gcutil --service_version="v1beta15"
   --project="accumulo-project" ssh  --zone="europe-west1-b"
   "hadoop-datanode"
```

    - For Windows:

```
python /usr/local/gcutil/gcutil.py
   --service_version="v1beta15" --project="accumulo-project"
   ssh  --zone="europe-west1-b" "hadoop-datanode"
```

- Follow the guidelines given in *Chapter 1, Building an Accumulo Cluster from Scratch*, to set up Hadoop (slave).

ZooKeeper

Create a single node for ZooKeeper.

- Create the ZooKeeper node:
  - For Linux:

```
/usr/local/gcutil/gcutil addinstance hadoop-zookeeper
   --machine_type=n1-standard-1 --image=centos-6-v20130731
   --zone=europe-west1-b
```

  - For Windows (Cygwin):

```
python /usr/local/gcutil/gcutil.py addinstance
   hadoop-zookeeper --machine_type=n1-standard-1
   --image=centos-6-v20130731 --zone=europe-west1-b
```

- Connect to the newly created ZooKeeper node:
  - For Linux:

```
/usr/local/gcutil/gcutil --service_version="v1beta15"
   --project="accumulo-project" ssh  --zone="europe-west1-b"
   "hadoop-zookeeper"
```

  - For Windows:

```
python /usr/local/gcutil/gcutil.py
   --service_version="v1beta15" --project="accumulo-project"
   ssh  --zone="europe-west1-b" "hadoop-zookeeper"
```

- Follow the guidelines given in *Chapter 1, Building an Accumulo Cluster from Scratch*, to set up ZooKeeper.

Accumulo

Create a single node for Accumulo both master and tablet server.

- Create the Accumulo node:
  - For Linux:

```
/usr/local/gcutil/gcutil addinstance accumulo-node1
   --machine_type=n1-standard-1 --image=centos-6-v20130731
   --zone=europe-west1-b
```

  - For Windows (Cygwin)

```
python /usr/local/gcutil/gcutil.py addinstance accumulo-
node1 --machine_type=n1-standard-1
--image=centos-6-v20130731 --zone=europe-west1-b
```

- Connect to the newly created Accumulo node:
  - For Linux:

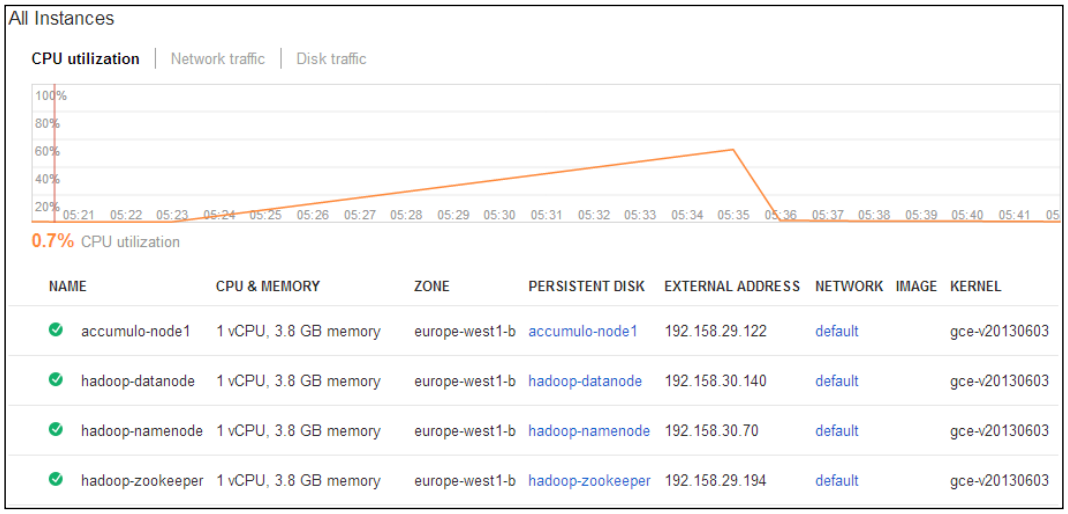
```
/usr/local/gcutil/gcutil --service_version="v1beta15"
   --project="accumulo-project" ssh --zone="europe-west1-b"
   "accumulo-node1"
```

  - For Windows:

```
python /usr/local/gcutil/gcutil --service_version="v1beta15"
   --project="accumulo-project" ssh  --zone="europe-west1-b"
   "accumulo-node1"
```

- Follow the guidelines given in *Chapter 1*, *Building an Accumulo Cluster from Scratch*, to set up Accumulo.

After the setup, you should see the following page in the Google Cloud Console, under **All Instances**:



Deleting the cluster

After we're done using the cluster, there is no reason to keep it around. By deleting the cluster, we are going to stop all the machines, remove all the data on scratch disks, and finally remove all the machines from the project. Scratch disk space is space tied to the life of an instance. That means when terminated, all scratch disk data is lost. In real scenarios store all data on persistent disks.

While deleting an instance, you will be asked if you want to delete the instance, and if you want to delete the persistent boot disk, answer both of those questions with a `Y`:

- Accumulo:
  - For Linux:

```
/usr/local/gcutil/gcutil deleteinstance "accumulo-node1"
   --zone=europe-west1-b
```

  - For Windows Cygwin:

```
python /usr/local/gcutil/gcutil.py deleteinstance "
   accumulo-node1" --zone=europe-west1-b
```

- ZooKeeper:
  - For Linux:

```
/usr/local/gcutil/gcutil deleteinstance "hadoop-zookeeper"
   --zone=europe-west1-b
```

  - For Windows Cygwin:

```
python /usr/local/gcutil/gcutil.py deleteinstance "
   hadoop-zookeeper" --zone=europe-west1-b
```

- Hadoop DataNode:
  - For Linux:

```
/usr/local/gcutil/gcutil deleteinstance "hadoop-datanode"
   --zone=europe-west1-b
```

  - For Windows Cygwin:

```
python /usr/local/gcutil/gcutil.py deleteinstance "
   hadoop-datanode" --zone=europe-west1-b
```

- Hadoop NameNode:
  - For Linux:

```
/usr/local/gcutil/gcutil deleteinstance "hadoop-namenode"
   --zone=europe-west1-b
```

  - For Windows Cygwin:

```
python /usr/local/gcutil/gcutil deleteinstance "
   hadoop-namenode" --zone=europe-west1-b
```

Rackspace

Rackspace has great support for Accumulo, Hadoop, and ZooKeeper. For Hadoop and ZooKeeper, there is a set of libraries called Apache Whirr that provides the ability to communicate with a large number of clouds by using low-level API libraries. This is exactly the same as for Amazon EC2. This section will focus on the difference between Amazon EC2 and Rackspace Cloud Services. Follow the Amazon EC2 steps with some minor changes as described in the *Configuration* section.

Configuration

Edit the `/usr/local/whirr/const/credentials` file and change these lines:

```
PROVIDER=clusterservers-us
IDENTITY=<your login from rackspace>
CREDENTIAL= <You API key>
```

For the `/usr/local/whirr/conf/cluster.properties` file, you need to change `whirr.provider` and provide ZooKeeper node, Hadoop NameNode, and Hadoop DataNode:

```
whirr.cluster-name=demo-cluster
whirr.instance-templates=1 zookeeper,1 hadoop-jobtracker+hadoop-
namenode,1 hadoop-datanode+hadoop-tasktracker
whirr.provider=cloudservers-us
whirr.private-key-file=${sys:user.home}/.ssh/id_rsa_whirr
whirr.public-key-file=${sys:user.home}/.ssh/id_rsa_whirr.pub
```

Setting up Accumulo on Rackspace cluster requires manual steps.

Log in to the Rackspace cloud console and create a new Linux machine using CentOS image, connect to it, and set up Accumulo manually as described in *Chapter 1*, *Building an Accumulo Cluster from Scratch*.

Network

A Rackspace cluster created with Whirr doesn't run behind a firewall. A firewall can be created manually by creating a new network, which is highly recommended to protect the cluster. More information on the topic of isolating a cloud network can be found at `http://www.rackspace.com/knowledge_center/article/create-an-isolated-cloud-network`.

Windows Azure

On February 1, 2010, Microsoft announced general availability of the Windows Azure cloud platform and infrastructure. Windows Azure supports both Microsoft Windows and Linux server operating systems. Windows Azure is a platform-closed source, but client SDK is open source.

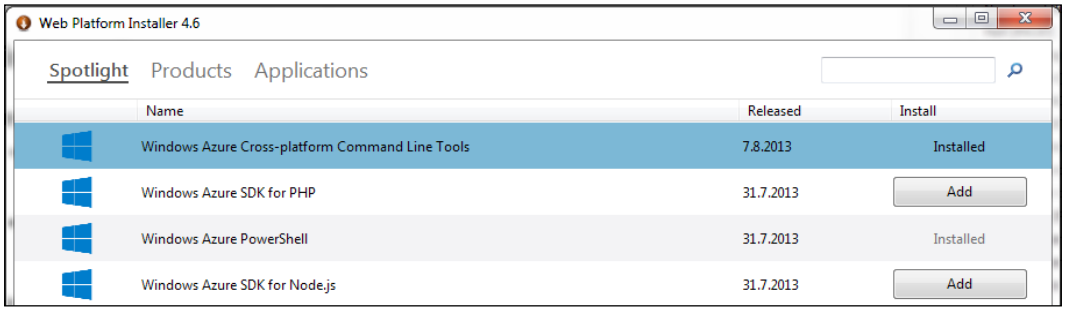
In the previous demonstrations, scripts have been used to create the cluster. But it can be as easy to use the interface provided by Windows Azure to get the same result if needed to create a small cluster.

Prerequisites

The prerequisites for Windows Azure are:

- Having a valid user to access **Windows Management Console**. Remember, billing is required to continue. Windows Azure Console is located at `https://manage.windowsazure.com`.
- If you want to use the command-line tool, install the command-line tools for Windows Azure from this location: `http://www.windowsazure.com/en-us/downloads/#cmd-line-tools`.

There are command-line tools for both Linux and Windows. Install both Windows Azure PowerShell and Cross-platform command-line interface. For Linux, you only need a command-line interface.



Creating the cluster

Windows Azure supports Windows Server 2012, OpenSUSE, SUSE Linux Enterprise Server, Ubuntu Server, and CentOS. In our example, we are going to use **Ubuntu Server 12.04 LTS**, but this changes on a regular basis, and you need to see what images are available before starting.

In this section, we are going to focus on the user interface of creating a cluster in Windows Azure. Using the command-line interface that is available for both Linux and Windows, it is very easy to accomplish the same task. Windows Azure PowerShell cmdlets need to be configured before use by following the guidelines in the article *Get Started with Window Azure Cmdlets* at `http://msdn.microsoft.com/en-us/library/windowsazure/jj554332.aspx`.

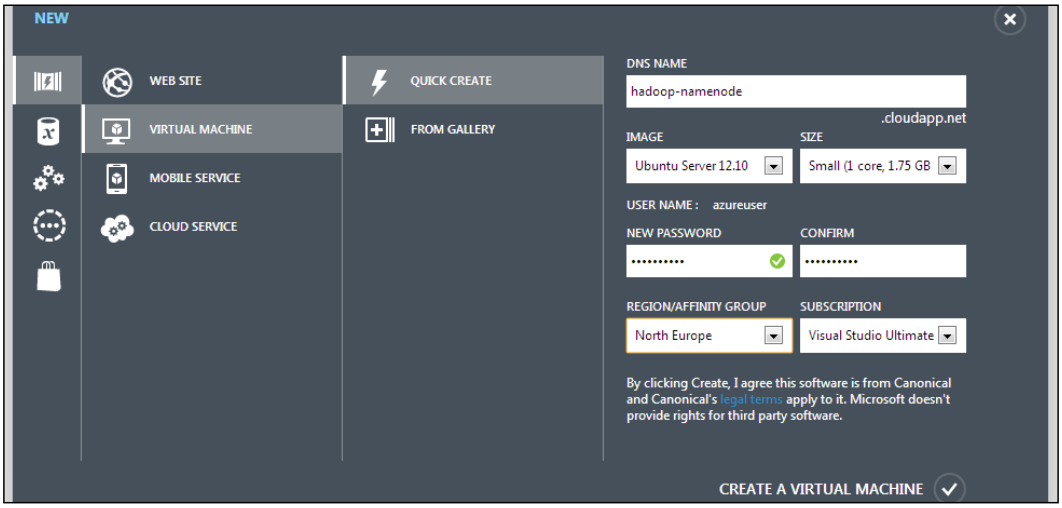
Creating a cluster involves four steps:

- Create and set up Hadoop NameNode. A new instance is created, and then Hadoop is set up and started as a master.
- Create and set up Hadoop DataNode. A new instance is created, and then Hadoop is set up and started as a slave.
- Create and set up a ZooKeeper node. A new instance is created, and then ZooKeeper is set up and started.
- Create and set up an Accumulo node. A new instance is created, and then Accumulo is set up and started.

Hadoop

For Hadoop we are going to create two nodes: NameNode and DataNode.

- Create the Hadoop NameNode by using the Windows Azure Management console. Create a new Linux virtual machine and give a meaningful name to the Hadoop NameNode. Nodes are available online (no firewall), so you need to pick a unique name for the Hadoop NameNode.



- Connect to the newly created Hadoop NameNode:
    - For Linux, use SSH
    - For Windows, use **PuTTY** to connect to the newly created computer
- Follow the guidelines given in the *Setting up Hadoop* section in *Chapter 1*, *Building an Accumulo Cluster from Scratch*, to set up Hadoop (master).
- Create the Hadoop DataNode:
    - By using the Window Azure Management Console, create a new Linux virtual machine and give a meaningful name to the Hadoop DataNode. Because nodes are available online (no firewall), you need to pick a unique name for the Hadoop DataNode.
- Connect to the newly created Hadoop DataNode:
    - For Linux, use SSH
    - For Windows, use PuTTY to connect to the newly created computer
- Follow the guidelines given in the *Setting up Hadoop* section in *Chapter 1*, *Building an Accumulo Cluster from Scratch*, to set up Hadoop (slave).

ZooKeeper

For ZooKeeper, we are going to create a single node.

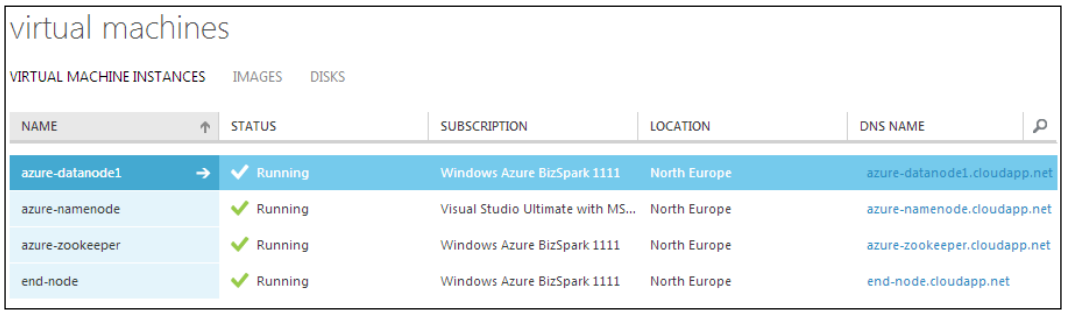
- Create the ZooKeeper node:
    - By using the Window Azure Management console, create a new Linux virtual machine and give a meaningful name to the ZooKeeper node. Because nodes are available online (no firewall), you need to pick a unique name for the ZooKeeper node.
- Connect to the newly created ZooKeeper node:
    - For Linux, use SSH
    - For Windows, use PuTTY to connect to the newly created computer
- Follow the guidelines given in the *Setting up ZooKeeper* section in *Chapter 1*, *Building an Accumulo Cluster from Scratch*, to set up ZooKeeper.

Accumulo

For Accumulo, we are going to create a single node.

- Create the Accumulo node:
  - By using the Window Azure Management console, create a new Linux virtual machine and give a meaningful name to the Accumulo node. Because nodes are available online (no firewall), you need to pick a unique name for the Accumulo node.
- Connect to the newly created Accumulo node:
  - For Linux, use SSH
  - For Windows, use PuTTY to connect to the newly created computer
- Follow the guidelines given in the *Setting up and configuring Accumulo* section in *Chapter 1*, *Building an Accumulo Cluster from Scratch*, to set up Accumulo.

After the setup, you should see the following page inside the Windows Azure console under **virtual machines**:



Deleting the cluster

After we have used our cluster, there is no reason to keep it around. By deleting the cluster, we are going to stop all machines, remove all scratch disk data, and finally remove all machines from the project.

While deleting an instance, you will be asked if you want to delete the instance, and if you want to delete the persistent boot disk. Answer both the questions with `Y`.

- **Accumulo**: In the Windows Azure Management console, select the Accumulo VM and then delete the VM
- **ZooKeeper**: In the Windows Azure Management console, select the ZooKeeper VM and then delete the VM

- **Hadoop DataNode**: In the Windows Azure Management console, select the Hadoop DataNode VM and then delete the VM
- **Hadoop NameNode**: In the Windows Azure Management console, select the Hadoop DataNode VM and then delete the VM

For more information about Ganglia, Graylog2 server, and Nagios, visit the following websites:

- **Ganglia**: `http://ganglia.info`
- **Graylog2 server**: `http://graylog2.org`
- **Nagios**: `http://www.nagios.org`

Summary

This chapter was about setting up Hadoop, ZooKeeper, and Accumulo on four different cloud platforms. There is a difference between those cloud platforms, but the steps required to set up an Accumulo cluster are very similar, meaning the only difference is the scripts used to automate the process. Even for Windows Azure, writing scripts for Windows or Linux is an easy task and is well documented.

In the next chapter, the focus is going to be on performance and how the optimal Hadoop cluster is configured, how much memory we really need, and CPU. Finally, we will look at configuring Accumulo systems to get the best performance.

4
Optimizing Accumulo Performance

This chapter will focus on how to optimize Accumulo's performance. As Accumulo uses Hadoop and ZooKeeper, we need to start with optimizing their basic performance.

A common setup of Accumulo consists of multiple nodes and usually a rack-mounted server system. Setting up a network for a multi-node Accumulo is very important, and will cause problem if it's not done correctly.

In this chapter, the focus is going to be on the configuration part to optimize Accumulo (and subsystems) for performance. It might sound strange to focus on the configuration in this chapter, but the best way to optimize Accumulo for performance is to know what kind of tasks it is used for and use that knowledge to tweak Accumulo.

The following topics will be covered in this chapter:

- Prerequisites
- Hadoop performance
- ZooKeeper performance
- Accumulo performance

Prerequisites

Before running any performance tuning on the Hadoop cluster, we need the following systems to be running and configured, as described in *Chapter 1, Building an Accumulo Cluster from Scratch*:

- **Ganglia** (`http://ganglia.info`): This is a scalable distributed monitoring system for high-performance computing systems that capture performance statistics.
- **Graylog2** (`http://graylog2.org`): This enables you to monitor an application's logs. This is an open source log management solution that stores your logs in ElasticSearch. The messages are accepted via TCP, UDP, or even AMQP, and are stored in MongoDB.
- **Nagios** (`http://www.nagios.org`): This is a powerful distributed monitoring system that reports system performance statistics.
- **Java profilers**
    - **JIP**: To know more about the Java Interactive Profiler, refer to `http://jiprof.sourceforge.net`
    - **Hprof**: This tool is shipped with the IBM SDK

  There is no reason to use both of these profilers; if you have the IBM SDK, then use Hprof, otherwise use JIP.

- **Linux performance tools**:
    - `netperf`: To know more about Netperf, refer to `http://www.netperf.org`
    - `htop`: This tool is an interactive process viewer
    - `iostat`: This tool is used for calculating CPU and I/O statistics for a network and filesystem

- **Windows performance tools**: The following tools are located in the Windows SDK:
    - **PsTools**: This tool is a part of Sysinternals tools. To know more about PsTools, refer to `http://live.sysinternals.com/`
    - **Resource Monitor**: This tool is accessed from the task manager

- **Hadoop tools**:
    - **Starfish Hadoop Log Analyzer** (`http://www.cu.duke.edu/starfish`): This is a good tool to indicate if Hadoop has been configured correctly for the current Map/Reduce job

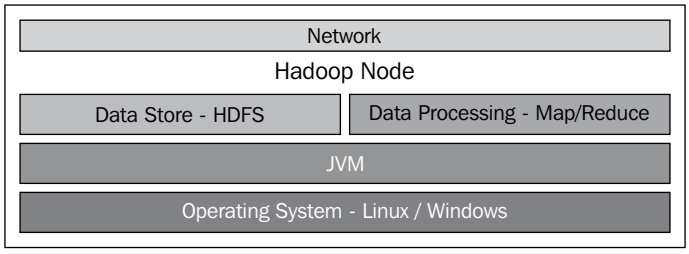
- **TestDFSIO**: This tool is located in the `/usr/local/hadoop/hadoop-*test*.jar` file. To learn how to use commands in Hadoop, refer to `http://hadoop.apache.org/docs/r1.2.1/commands_manual.html`
- **TeraSort**: This is located in the `/usr/local/hadoop/hadoop-*example*.jar` file

In most cases, we are focusing on network utilization, CPU utilization, and memory usage. Ganglia and Nagios have excellent support. Remember that this is all about getting data to spot performance bottlenecks, and not about the tools to use. This is just a limited list, and better tools might be available.

Hadoop performance

There are a few important items that you need to be aware of while configuring Hadoop for performance. Hadoop is based on and written in Java, as a distributed computing framework. Since Hadoop is highly scalable, you need to consider many options depending on the requirements you have, regarding the performance and stability of your setup.

The following image gives the baseline for a better understanding of how Hadoop relies on the JVM and operating system. Knowing how to tweak JVM for performance is as important as knowing how to tweak the operating system for performance.



Baseline

To establish a baseline, we need to perform a stress test to verify that the setup is according to the requirement. There are many ways in which a baseline can be established, but in our case, to ensure that we have enough disk storage, we need to configure the Java heap size to be around 70 percent (or up to, but not more) of the available memory, so that there will still be enough memory left for the operating system (Linux or Windows), to perform operations without swapping. We need to configure Map/Reduce slots to utilize the CPU.

What is usually configured is a Java heap size, and we are aiming for using around 70 percent of the available memory. If too much memory is used, the operating system starts swapping.

Tuning

The default values in Hadoop are usually too low for any real workload; increase them to increase performance.

Using the Java profiler, we can see GC patterns, heap usage, and heap lock to find the correct value. For JVM, the following are a few points to consider :

- **Biased locking**: This improves performance in most cases
- **Compressed pointers**: By default, this is on and will reduce memory
- **AggressiveOpts**: This option is used for point performance compiler optimizations
- **Code Cache**: Try to increase JVM Code Cache if you are running out of code cache

Tuning parameters for mapred-default.xml

This section shows examples of configuration parameters that are useful when configuring Accumulo, Hadoop, and ZooKeeper clusters. There is no way of going around optimizing performance using the tools stated in the *Prerequisites* section of this chapter to get the best result.

Remember to keep the Reducer to Mapper ratio as 3:4. The basic rule is to have 1 core to handle one slot, but it has been shown to work for up to 1.2 cores to handle one slot.

The changes recommended in `mapred-site.xml` are shown in the following table:

Configuration parameter	Default value	Comment
`mapreduce.task.timeout`	`600000`	Its default value is 600 seconds. This depends on the job; it might be needed to increase this value.
`mapred.map.tasks`	`2`	This parameter refers to the number of map tasks per job. Its value is ignored if `mapred.job.tracker` is local.
`mapred.tasktracker.map.tasks.maximum`	`2`	This parameter refers to the maximum number of tasks that a task tracker can run simultaneously.

Configuration parameter	Default value	Comment
`mapred.reduce.tasks`	`1`	The number of reduce tasks per job. The default value is ignored when `mapred.job.tracker` is local.
`mapred.tasktracker. reduce.tasks.maximum`	`2`	This parameter is configured between half the number of cores per node and two times the number of cores.
`mapred.map.child.java. opts`		This parameter starts with the –Xmx512M value.
`mapred.reduce.child. java.opts`		This parameter starts with the –Xmx1024M value.
`io.sort.mb`	`100`	This parameter is configured to consume up to 200 MB or 70 percent of the Java heap size. If it is done right, it's possible to reduce the number of spills.
`fs.inmemory.size.mb`	`100`	This parameter is configured to consume up to 200 MB or 70 percent of the Java heap size.
`io.sort.factor`	`100`	This parameter finds the number of seeks being done when merging the files. If too high, then the seek cost on the disk will exceed the savings. You can use Starfish to find the correct value.
`mapred.reduce.parallel. copies`	`5`	To configure this parameter for bigger clusters, set this to a higher number.

HDFS

Hadoop supports compression that is very useful when there is a need for processing large data but there is only small cluster running and available. Using compression is always good way to extend small cluster capabilities.

Using Hadoop support for compression is an option worth exploring. Often, better compression could mean more CPU cycles, and if the cluster is already doing a lot of CPU bound work, it is a good idea to use codec that uses fewer CPU cycles and even skip using compression. The **Lempel–Ziv–Oberhumer** (**LZO**) is a lossless data compression algorithm, and it has been proven in research to be best suited for performance.

Hadoop had, by default, few compression formats, as shown in the following table:

Compression formats	Hadoop compression codec
DEFLATE	`org.apache.hadoop.io.compress.DefaultCodec`
gzip	`org.apache.hadoop.io.compress.GzipCodec`
LZO	`com.hadoop.compression.lzo.LzopCodec`

For more information about the LZO lossless data compression algorithm, refer to the following link:

`http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Oberhumer`

In order to set up LZO to compress intermediate map output, add the following code in `mapred-site.xml`:

```
<property>
  <name>mapreduce.map.output.compress</name>
  <value>true</value>
</property>
<property>
  <name>mapreduce.map.output.compress.codec</name>
  <value>com.hadoop.compression.lzo.LzoCodec</value>
</property>
```

Tuning parameters for mapred-site.xml

This section includes examples of configuration parameters that can be useful when configuring compression and block size.

The changes recommended in `mapred-site.xml` are shown in the following table:

Configuration parameter	Default value	Comment
`mapred.compress.map.output`		This parameter performs data compression between the Mapper and the Reducer.
`mapred.map.output.compression.codec`		This parameter is a compression codec between the Mapper and the Reducer. For performance, use LZO.
`mapred.output.compression.type`	`RECORD`	For this parameter, instead of compressing a single `RECORD`, use `BLOCK`, which compresses a group of records.

Configuration parameter	Default value	Comment
`mapred.output.compression.codec`		This parameter is a compression codec between the Mapper and the Reducer. For performance, use LZO.
`mapred.min.split.size`		This parameter is the smallest number that has a valid size in bytes for a file split. The best way of setting the input split is by using the HDFS block size; this also applies to `mapred.max.split.size`.
`mapred.max.split.size`		This parameter is the largest number that has a valid size in bytes for a file split.

Tuning parameters for hdfs-site.xml

The change recommended in `hdfs-site.xml` is shown in the following table:

Configuration parameter	Default value	Comment
`dfs.block.size`	`64 M`	Here you can use larger block sizes such as 128 MB, or even 256 MB, if using larger files.

For more information about real-world Hadoop cluster configuration, please visit `http://hadoop.apache.org/docs/stable/cluster_setup.html`.

ZooKeeper performance

To be able to understand and spot problems in ZooKeeper cluster, we need to establish a baseline. On the ZooKeeper website is the *Service Latency Overview* page which is a good starting ground to establish the baseline:

`http://wiki.apache.org/hadoop/ZooKeeper/ServiceLatencyOverview`

Because every node in ZooKeeper has to vote, it will cause an increased cost in writing operations when the cluster size increases. The flipside is, if too many clients connect to the same node, then it's possible for that node to go down. Then you will only have three ZooKeeper servers running (1 leader and 2 followers), and after one is down, only one follower will still be alive, and if all the clients connect to it, they might take it down. When you come across this problem, you need to change the setup of the ZooKeeper cluster.

To verify if the setup of ZooKeeper is correct, it is recommended to run a smoke test and a latencies test. ZooKeeper uses heart-beating between clients and servers, which causes ZooKeeper to handle network and system latencies poorly. It is a good choice to run smoke and latency tests using the `zk-smoketest` project on Github which can be found at `https://github.com/phunt/zk-smoketest`.

For JVM configurations, use the same configuration for the Java max heap size as for Hadoop. Never use more that 70 percent of the available memory.

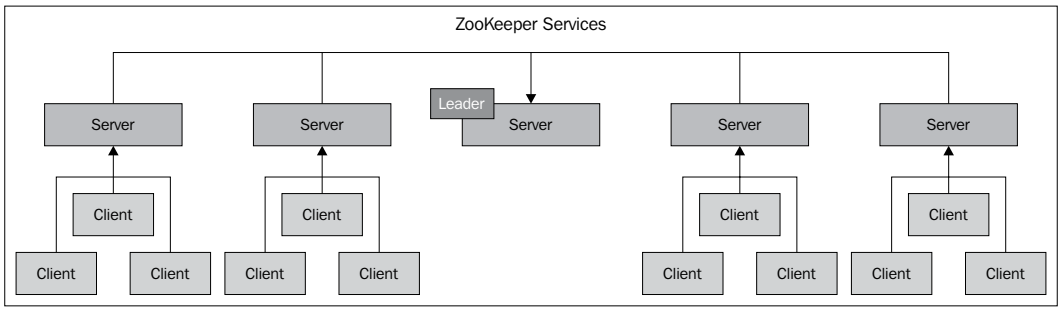
Use Graylog2 to monitor the logfiles and Ganglia to monitor each ZooKeeper server.

ZooKeeper overview

The number of nodes in the ZooKeeper setup is always an odd number. Adding too many ZooKeeper nodes to the cluster doesn't improve performance. ZooKeeper has a few guarantees:

- **Sequential consistency**: The consistency order is preserved
- **Atomicity**: There are no partial results; only succeed or fail
- **Single system image**: All nodes will provide the same answer to the client
- **Reliability**: All updates will persist

The following image shows typical setup of five ZooKeeper nodes where one is leader and the rest are followers. The leader is chosen at startup.



Accumulo performance

Accumulo uses Hadoop HDFS as a filesystem and ZooKeeper as a coordinator. If the problem isn't in Hadoop or ZooKeeper, we need to look at Accumulo. By now, you should be monitoring the logs with Graylog2 or similar, and monitoring the Accumulo TabletServer(s) and the Accumulo Master with Ganglia.

Accumulo has fewer configuration values than Hadoop, but the same rule applies to the configuration of JVM.

Tuning parameters for accumulo-site.xml

The changes recommended in `accumulo-site.xml` are shown in the following table:

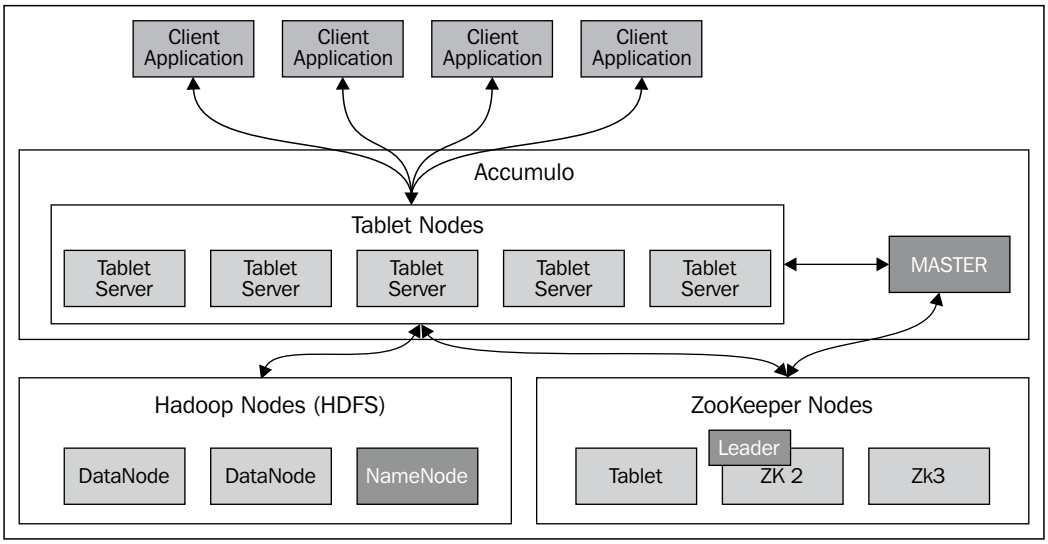
Configuration parameter	Default value	Comment
`instance.zookeeper.host`	`localhost:2181`	Here, ZooKeeper servers are separated by a comma.
`instance.dfs.dir`	`/user/accumulo/accumulo`	This refers to the location where Accumulo stores files in Hadoop.
`logger.dir.walog`	`walogs`	This indicates the write-ahead logs to the local system.
`instance.secret`		This has to be the same value for all servers.
`tserver.memory.maps.max`	`80 M`	This refers to the maximum TabletServer memory to be used (bigger is better), considering a node with minimal 16 GB
`tserver.cache.data.size`	`7M`	This refers to the cache data size on the TabletServer. For a node with 16 GB, consider 2 GB, but this depends on the amount of data you are storing.
`tserver.cache.index.size`	`20 M`	This refers to the cache index size on the TabletServer. For a node with 16 GB, consider 1 GB, but this depends on the amount of data you are storing.
`tserver.walog.max.size`	`100 M`	This depends on the size of the disk on the node (I have been using 8 GB when running two TabletServers).

While experiencing a slow response from Accumulo, the best place to start is the *status* page (`http://localhost:50095/status`) to find performance bottlenecks.

Accumulo overview

The following figure tries to give an example of a simple Accumulo cluster. In the cluster, we have five Accumulo **TabletServer** nodes to handle requests from multiple clients that are connected to them, and one Accumulo **MASTER** node.

The **Tablet Nodes** connect to Hadoop and use the HDFS, but use ZooKeeper to coordinate. This setup is very good to practice using Accumulo, and optimizing its performance. This setup is easy to do using a cloud provider or a local virtualize environment.



Accumulo's performance summary

To activate the highest ingest performance, a focus on both the ingestion and query components is required. By using the Accumulo status page, you can get a rough indication if there are any performance bottlenecks, and compare CPU, network, and memory numbers from Ganglia. But usually, you can know upfront. It is possible to scale Accumulo up to the point where either the aggregate I/O of TabletServer(s) or the total network bandwidth capacity is reached.

By monitoring the Accumulo graphs on the status page, and also by using the Starfish Hadoop Log Analyzer tool, configuring Accumulo can be an easy task.

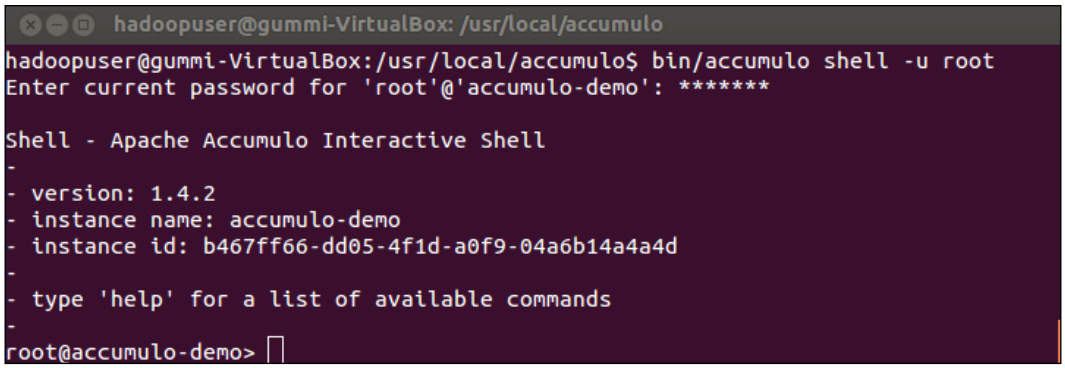
Tables

While creating a new table, it will be on a single tablet; this is the default behavior. For performance, this would not be good practice, but when more data is added to the table, the table is automatically split to many tablets. While running 10 TabletServer(s), it will take some time until the table has been split into tablets on every TabletServer. If you know that the table will grow, and you want to take advantage of the cluster setup and get the most out of parallelism in the cluster, you need to think about *pre-splitting* the table.

Manually, pre-splitting is an easy task and can be done through the Accumulo shell using the following steps:

1. Log into the Accumulo shell using the following command:

   ```
   cd /usr/local/accumulo
   ./bin/accumulo shell -u root
   ```



2. Create a new table using the Accumulo shell command:

   ```
   root@accumulo-demo> createtable mydemotable1
   ```

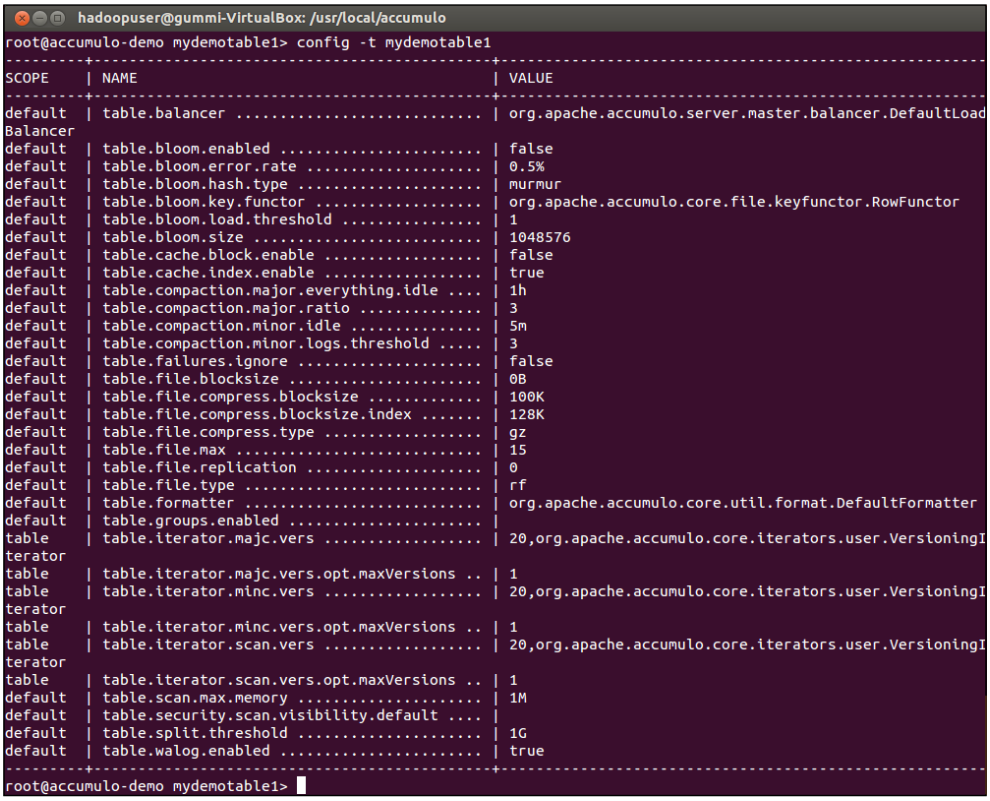
3. Pre-splitting tables is done with the following Accumulo shell command:

   ```
   root@accumulo-demo> addsplits –s /local_splitfile – t
   mydemotable1
   ```

4. The table configuration values can be obtained using the Accumulo shell command:

   ```
   root@accumulo-demo> config -t mydemotable1
   ```

5. Calling `config` for any given table gives detailed information. Notice that the default value for `table.split.threshold` is `1G`, which is very useful when knowing the size of the data.



Accumulo has many useful commands, and for more information on *Accumulo Shell Command*, go to *Appendix A*, *Accumulo Command Reference*.

Comparing bulk ingest versus batch write

Accumulo provides support for importing files by using the Accumulo shell command `importdirectory`, and multiple files can be written to the existing tables. Clients can use `BatchWriter` via ingestion. Using `BatchWriter`, a large amount of data can be formatted as Accumulo expects. In many cases, it is faster to use bulk ingest, import files directly into Accumulo instead of using `BatchWriter`, and client writing data to Accumulo using the API.

To get splits on any given table, use the following Accumulo shell command:

```
root@accumulo-demo> getsplits -t mydemotable1
```

Accumulo examples

Accumulo has an excellent documentation that will give you an idea on how you should set up each node in the cluster. Please read more about it by visiting the following link:

```
http://localhost:50095/docs
```

Accumulo ships with many useful examples in order to understand the different aspects of operations and performance. The following list has examples that will be useful to get to know Accumulo and understand the best way to solve different use cases:

- **batch**: You can refer to `http://accumulo.apache.org/1.5/examples/batch.html` to know how batch writer and batch scanner are used
- **bloom**: You can refer to `http://accumulo.apache.org/1.5/examples/bloom.html`, which explains how the Bloom filter is used to increase query performance
- **bulk ingest**: You can refer to `http://accumulo.apache.org/1.5/examples/bulkIngest.html` to understand how to use Map/Reduce jobs to perform ingest bulk data
- **classpath**: You can refer to `http://accumulo.apache.org/1.5/examples/classpath.html` to know how a classpath is used for each table
- **client**: You can refer to `http://accumulo.apache.org/1.5/examples/client.html` to know how to use table operations
- **combiner**: You can refer to `http://accumulo.apache.org/1.5/examples/combiner.html` to know how to use StatsCombiner for min, max, sum, and count
- **constraints**: You can refer to `http://accumulo.apache.org/1.5/examples/constraints.html` to know how to use constraints with tables
- **dirlist**: You can refer to `http://accumulo.apache.org/1.5/examples/dirlist.html` to know how a filesystem's information is stored
- **export**: You can refer to `http://accumulo.apache.org/1.5/examples/export.html` to know how tables can be imported or exported
- **file data**: You can refer to `http://accumulo.apache.org/1.5/examples/filedata.html` to know how file data is stored
- **filter**: You can refer to `http://accumulo.apache.org/1.5/examples/filter.html` to know how 30 second old records are removed
- **hello world**: You can refer to `http://accumulo.apache.org/1.5/examples/helloworld.html` to know how Map/Reduce jobs can store data from inside and outside

- **isolation**: You can refer to `http://accumulo.apache.org/1.5/examples/isolation.html` to know how the isolation scanner is used
- **mapreduce**: You can refer to `http://accumulo.apache.org/1.5/examples/mapred.html` to know how Map/Reduce can read and write
- **maxmutation**: You can refer to `http://accumulo.apache.org/1.5/examples/maxmutation.html` to know how to avoid running out of memory
- **regex**: You can refer to `http://accumulo.apache.org/1.5/examples/regex.html` to know how to use regular expressions
- **rowhash**: You can refer to `http://accumulo.apache.org/1.5/examples/rowhash.html` to know how to read a table and write a row in a table
- **shar**: You can refer to `http://accumulo.apache.org/1.5/examples/shard.html` to know how to use the intersection iterator
- **table-to-file**: You can refer to `http://accumulo.apache.org/1.5/examples/tabletofile.html` to know how to read a table and write to HDFS
- **terasort**: You can refer to `http://accumulo.apache.org/1.5/examples/terasort.html` to know how to use random data and sorting
- **visibility**: You can refer to `http://accumulo.apache.org/1.5/examples/visibility.html` to know how to use visibility

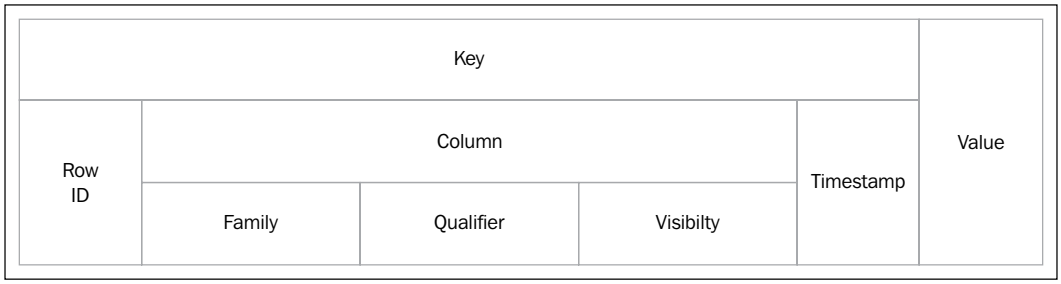
Summary

The most important aspect when optimizing a cluster is disk and memory usage, which is divided between nodes over the cluster. Understanding the configuration parameters for Hadoop, ZooKeeper, and Accumulo is vital to be able to get the best performance out of the cluster.

In the next chapter, we will go through the security aspect of Accumulo. We will learn how Accumulo uses cell-level security to gain full control over the visibility of every cell for every table, and the following chapter will give us the answer to questions such as how Accumulo is able to secure information sharing and information multitenancy.

5
Security

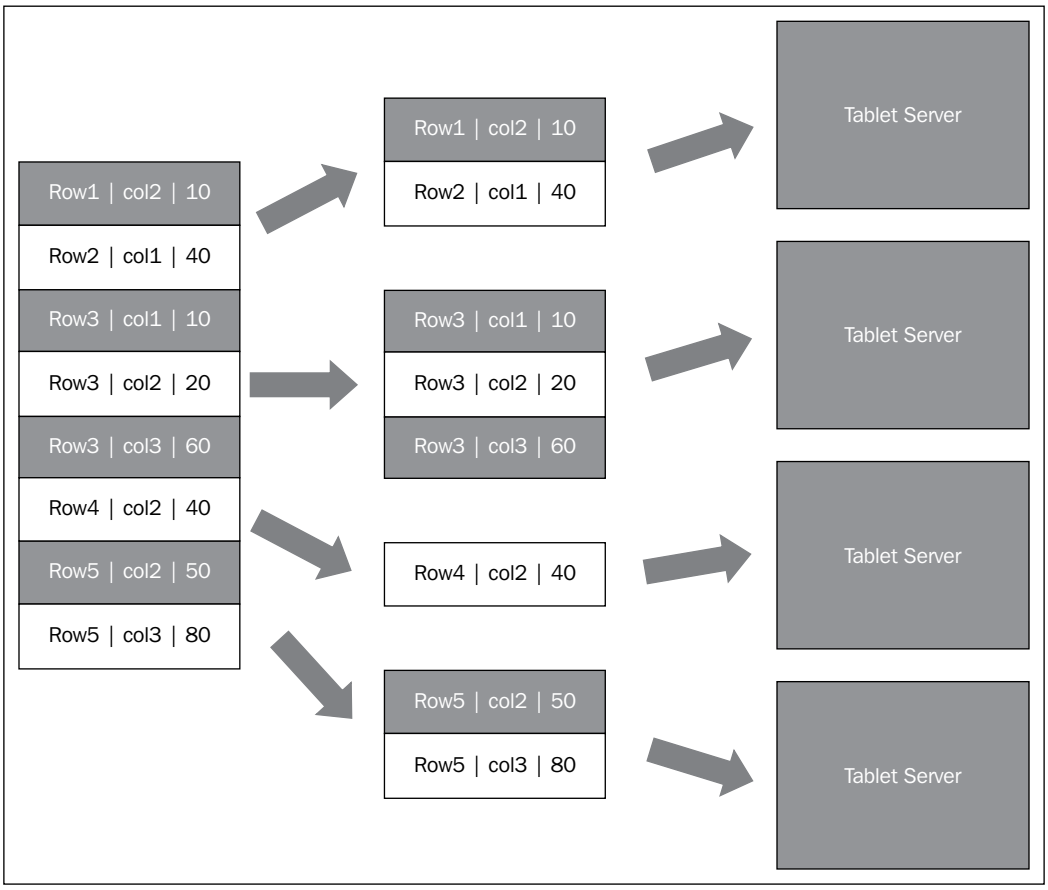
Accumulo is designed for fine-grained security that normal database systems don't support. In the relational database world, the normal rule is that if you are allowed to query a table, you are going to be able to see all the rows in that table. Accumulo is designed to extend BigTable and fully supports cell-level security. Accumulo is a Key-Value database where one data row or Key-Value pair is composed of the following elements:



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

The data is distributed by partitioning the table into tablet servers, as follows:



You can group table security into four groups:

- **Table-level security**: Using this feature, it is possible to control the insert, update, delete, and select operations for the entire table.
- **Column-level security**: The idea is to control columns in the table. It is useful when it's needed to freeze columns in the table.
- **Row-level security**: This feature is useful if you want to control what rows are accessible by some business logic.
- **Cell-level security**: Accumulo has a powerful security feature called cell-level security. Using this feature gives you full control over the visibility of every cell for every table.

In this chapter, the focus is going to be on cell-level security.

These are the topics we'll cover in this chapter:

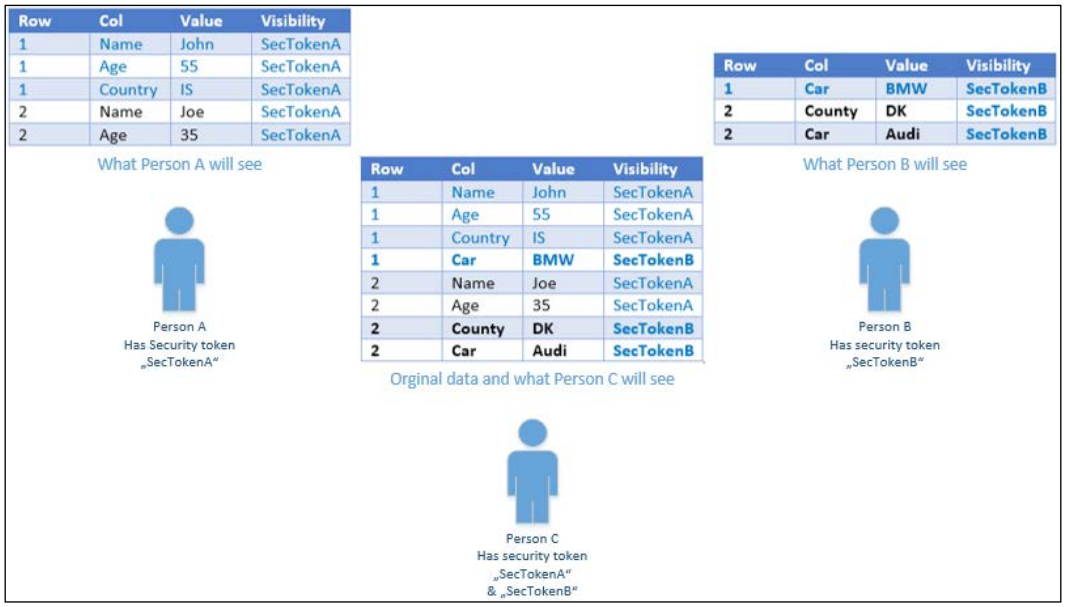
- Visibility
- Security expression
- Authorization
- User authorizations
- Handling secure authorization
- Query Services Layer

Visibility

Visibility is an element for Accumulo, for every Key-Value row. Cell-level security is accomplished by controlling the visibility for every Key-Value pair. The flexible security Accumulo provides is better than what relational databases provide for multi-tenant scenarios, meaning when fine-grained security restrictions are required for single table.

In the following example, there are three persons with different authorization tokens. **Person A** has **SecTokenA**, **Person B** has **SecTokenB**, and finally **Person C** has both **SecTokenA** and **SecTokenB**. When Person A queries the table in the middle, that person will only see rows where the visibility is set to the authorization token it has, that is, SecTokenA. The same applies to Person B, who only sees rows where the visibility is SecTokenB. Finally, Person C with both authorization tokens, SecTokenA and SecTokenB, will see all of the rows.

The authorization token is a string, and it is important to use descriptive names. Compared to Active Directory, the authorization tokens are similar to AD groups.



Creating an Accumulo user

Before we start, let's examine the process of creating a user in Accumulo and what rights the new user has when created. While doing this, it is possible to use the `-s` switch or `-scan-authorizations`, followed by authorizations separated by comma(s) to set the security token.

Use the following command to start a new Accumulo shell:

```
/usr/local/accumulo/bin/accumulo shell -u root
```

Use the following command to create a user in Accumulo:

```
root@accumulo-demo> createuser accumulouserA
```

Creating tables in Accumulo

Creating tables in Accumulo that are Key-Value tables, is done by simply using the `createtable` command with the table name. In normal relational databases, tables are created by listing out columns; however, in Key-Value databases, columns are created when data is written. This difference makes Accumulo more flexible. The only security concern is to add an access control label to each Key-Value pair.

To create a new table as root, use the following command:

```
root@accumulo-demo> createtable mydemotable2
```

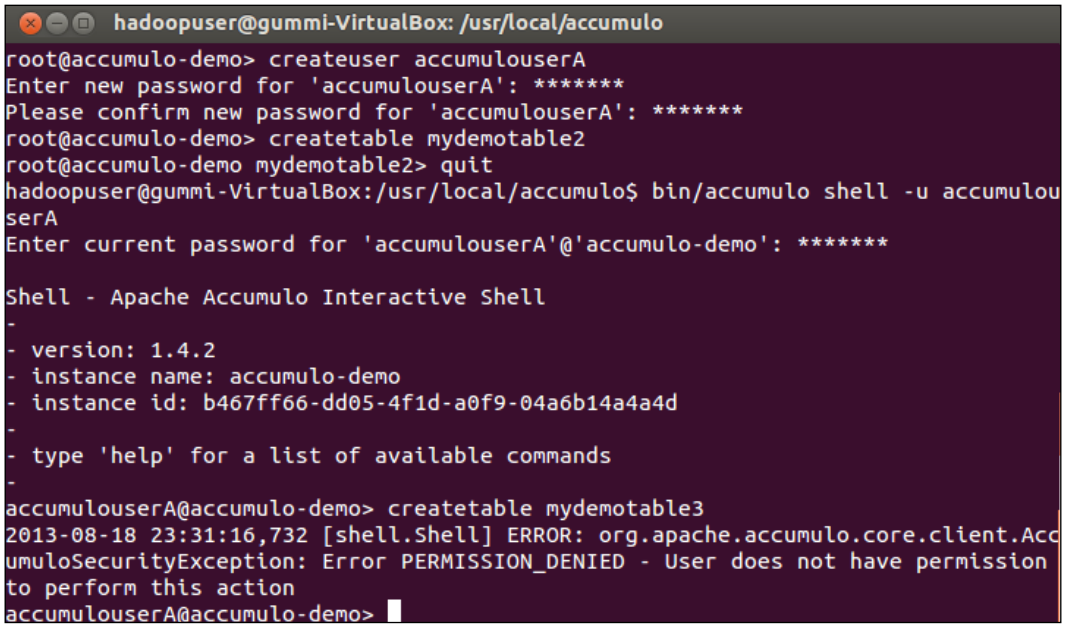
How does visibility work?

Visibility is all about who has access to the data. It works by using fine-grained tagging of every row (Key-Value pair), with an access control label.

Root has the right to create a table. Let's log out and log in as the new user that we created using the following:

```
root@accumulo-demo>quit
/usr/local/accumulo/bin/accumulo shell -u accumulouserA
accumulouserA@accumulo-demo>create mydemotable3
```

Now, you get the `User does not have permission to perform this action` error, as shown in the following screenshot:



To be able to create a table in Accumulo as `accumulouserA`, you need to be granted with `System.CREATE_TABLE` permission.

So, start a new Accumulo shell using the following command:

```
/usr/local/accumulo/bin/accumulo shell -u root
```

Grant the `CREATE_TABLE` permission to `accumulouserA` using the following command:

```
root@accumulo-demo> grant -s System.CREATE_TABLE -u accumulouserA
```

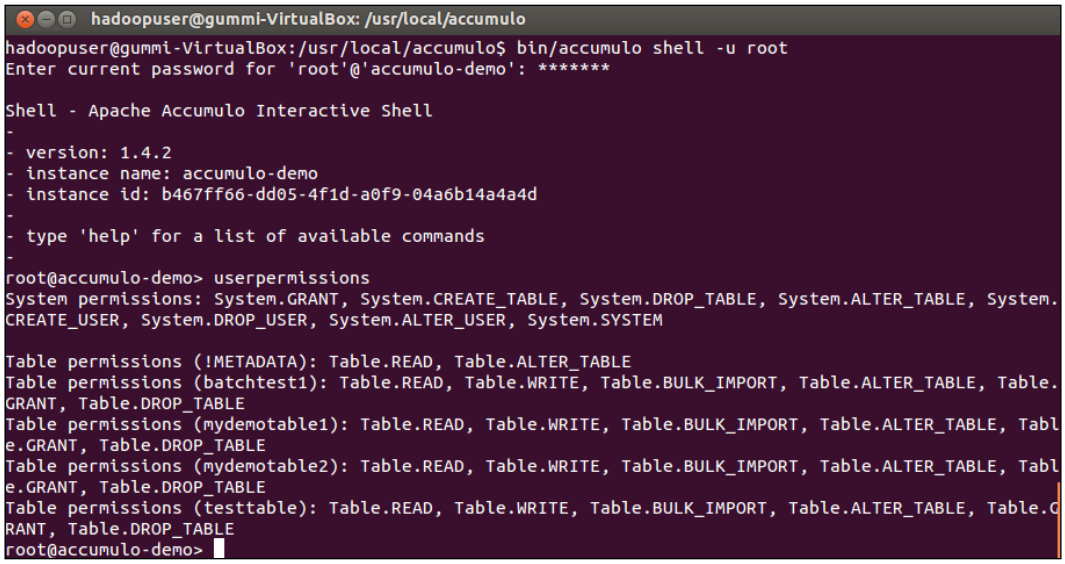
Before you grant permission to a user, it's a good rule to get a list of access permissions for that user.

Ask for permissions for the current logged-in user by using the following command:

```
root@accumulo-demo> userpermissions
```

Or ask for permissions for a specified user by using the following command:

```
root@accumulo-demo> userpermissions -u accumulouserA
```



Visibility of a row is controlled via Boolean `&` and `|` operators, where you can combine both of them in a sentence.

Let's look at the following example of inserting rows with different visibility tokens. The steps taken are:

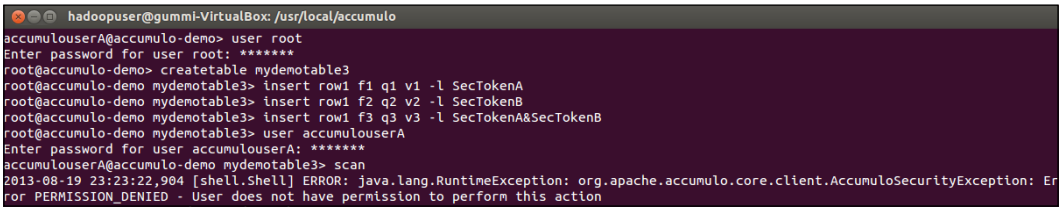
1. Create a table.
2. Insert <row> <colfamily> <colqualifier> <value> with the authorization label `SecTokenA`.

3. Insert <row> <colfamily> <colqualifier> <value> with the authorization label `SecTokenB`.
4. Insert <row> <colfamily> <colqualifier> <value> with the authorization label `SecTokenA&SecTokenB`. There's support for the use of boolean algebra for an authorization label.
5. Switch from `root` to `accumulouserA`.
6. Scan the table and the following will be shown:

```
root@accumulo-demo> createtable mydemotable3
root@accumulo-demo mydemotable3> insert row1 f1 q1 v1 -l SecTokenA
root@accumulo-demo mydemotable3> insert row1 f2 q2 v2 -l SecTokenB
root@accumulo-demo mydemotable3> insert row1 f3 q3 v3 -l
   SecTokenA&SecTokenB
root@accumulo-demo mydemotable3> user accumulouserA
Enter password for user accumulouserA: *******
accumulouserA@accumulo-demo mydemotable3> scan
2013-08-19 23:23:22,904 [shell.Shell] ERROR:
   java.lang.RuntimeException: org.apache.accumulo.core.client.
AccumuloSecurityException: Error
   PERMISSION_DENIED - User does not have permission to perform
this
   action
```

As user authorizations are a set of authorization tokens—and by default the user authorizations set is empty—you need to change the authorizations set with the user that has rights. Also, the user has to have access to the table.

The following screenshot shows what happens when a user tries to scan a table and doesn't have access to it. This isn't related to the security tokens; it's related to the fact that the user doesn't have an access to the table.



Let's examine what happens if we create few cells and then try to scan as follows:

```
root@accumulo-demo mydemotable3> insert row1 f1 q1 v1 -l SecTokenA
root@accumulo-demo mydemotable3> insert row1 f2 q2 v2 -l SecTokenB
```

```
root@accumulo-demo mydemotable3> insert row1 f3 q3 v3 -l
SecTokenA&SecTokenB
root@accumulo-demo mydemotable3> scan
```

You are going to get an empty list because by default, the user authorizations set is empty even for the `root` user.

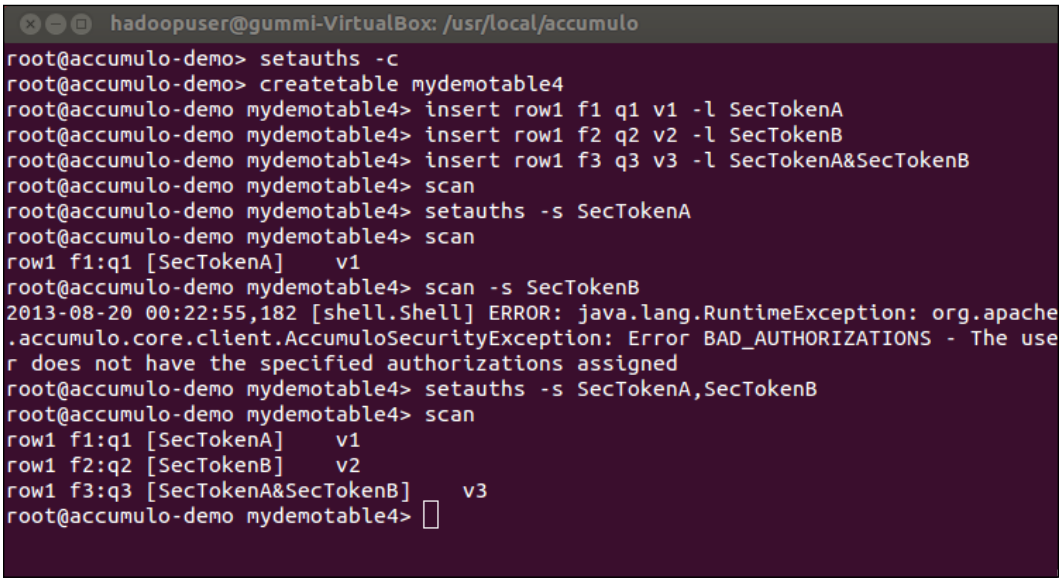
If we change the user's authorizations set to the token string `SecTokenA`, we will only get one row in the result set as follows:

```
root@accumulo-demo mydemotable3> setauths -s SecTokenA
root@accumulo-demo mydemotable3> scan
row1 f1:q1 [SecTokenA]    v1
```

If we try to scan the table using an authorizations set that the user doesn't have, we will get an error as follows:

```
root@accumulo-demo mydemotable3> scan -s SecTokenB
2013-08-19 23:45:24,709 [shell.Shell] ERROR:
   java.lang.RuntimeException:
   org.apache.accumulo.core.client.AccumuloSecurityException:
   Error BAD_AUTHORIZATIONS - The user does not have the specified
   authorizations assigned
```

Finally, set the authorizations set to both `SecTokenA` and `SecTokenB`; now we are getting to the result we expected, as shown in the following screenshot:



Security expression

As we saw in the previous example, security labels are a set of tokens the user chooses to use. The tokens are a set of ASCII characters (arbitrary strings). In the previous example, we used the logical AND operator to join two authorization tokens.

Examples of usage are:

- **AND**: An example of this is `SecTokenA&SecTokenB`, where the user needs both `SecTokenA` and `SecTokenB` to be able to see the cell
- **OR**: An example of this is `SecTokenA|SecTokenB`, where the user only needs one token, `SecTokenA` or `SecTokenB`, to be able to see the cell
- **AND, OR**: An example of this is `(SecTokenA&SecTokenB)|SecTokenC`, where the user only needs either `SecTokenA` and `SecTokenB` or `SecTokenC`

More complex rules can be created using Boolean `AND` (`&`) and `OR` (`|`) combinations.

It is always a good idea to create company definitions for the naming of security labels and user. Often the terms of user roles are used, such as `power-user` and `admin`.

Writing a Java client

Writing a Java client and setting different visibility for two different values for the same row ID is a fairly simple task. The code to create it is as follows:

```
import org.apache.accumulo.core.client.AccumuloException;
import org.apache.accumulo.core.client.AccumuloSecurityException;
import org.apache.accumulo.core.client.BatchWriter;
import org.apache.accumulo.core.client.Connector;
import org.apache.accumulo.core.client.Instance;
import org.apache.accumulo.core.client.TableExistsException;
import org.apache.accumulo.core.client.TableNotFoundException;
import org.apache.accumulo.core.client.ZooKeeperInstance;
import org.apache.accumulo.core.data.Mutation;
import org.apache.accumulo.core.data.Value;
import org.apache.accumulo.core.security.ColumnVisibility;
import org.apache.hadoop.io.Text;

public class AccumuloDemo2 {

   public static void main(String[] args)
      throws AccumuloException,
      AccumuloSecurityException,
      TableNotFoundException,
      TableExistsException{
```

```
        // Constants
        String instanceName = "accumulo-demo";
        String zooServers = "zooList";
        String userName = "<change>";
        String password = "<change>";

        // Connect
        Instance inst = new ZooKeeperInstance(instanceName,
                                              zooServers);
        Connector conn = inst.getConnector(userName, password);

        // Use batch writer to write demo data
        BatchWriter bw = conn.createBatchWriter("demotable",
                                                1000000, 60000,
                                                2);
        // Create two columns for the same row
        Text rowID = new Text("row2");
        long timestamp = System.currentTimeMillis();
        // Visibility "SecTokenA"
        ColumnVisibility colVisA =
                              new ColumnVisibility("SecTokenA");
        // Visibility "SecTokenB"
        ColumnVisibility colVisB =
                              new ColumnVisibility("SecTokenB");
        // Create column(family, qualifier) and value
        Text colFam1 = new Text("colFam");
        Text colQual1 = new Text("colQual");
        Value value1 = new Value("some-value".getBytes());
        // create first new mutation and add rowID, colFam,
        // colQual, value
        Mutation mutation1 = new Mutation(rowID);
        mutation1.put(colFam1, colQual1, colVisA, timestamp,
                      value1);
        bw.addMutation(mutation1);
        // Create column(family, qualifier) and value
        Text colFam2 = new Text("colFam");
        Text colQual2 = new Text("colQual");
        Value value2 = new Value("some-value".getBytes());
        // create second new mutation and add rowID, colFam,
        // colQual, value
        Mutation mutation2 = new Mutation(rowID);
        mutation2.put(colFam2, colQual2, colVisB, timestamp,
                      value2);
        // add the mutation to the batch writer
        bw.addMutation(mutation2);
        // close the batch writer
        bw.close();
    }
}
```

Authorization

Most clients are written in Java, but you can use the proxy API to interact with Accumulo in other languages. As we saw in the preceding example, when using security tokens we need to pass them to the Accumulo instance. Then, write the Java client and create the connection (a Java code), as follows:

```
String inName = "accumulo-demo";
String zooKeeperServers = "zkServer1,zkServer2,zkServer3";
Instance zkIn = new ZooKeeperInstance(inName, zooKeeperServers);
Connector conn = zkInstance.getConnector("myuser", "password");
```

Create authorization (a Java code), as follows:

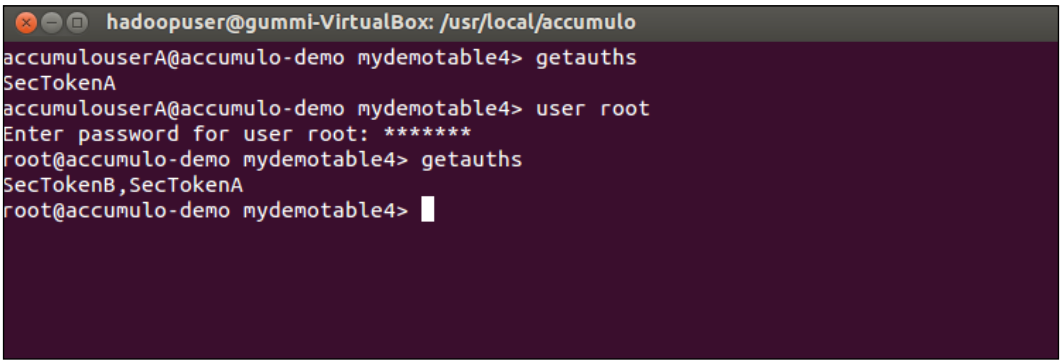
```
Authorization auths = new Authorization("SecTokenA", "SecTokenB");
Scanner s = conn.createScanner("mydemotable4", auths);
```

There are many interesting projects supporting Accumulo, and there is even a Python client library from Apache Accumulo, available at `https://github.com/accumulo/pyaccumulo`.

User authorizations

Use the Accumulo shell to change the security label for users. There are two commands that you need to use:

- `setauths`: It sets authorization tokens for the current user or a specific user when using the `-u` flag
- `getauths`: It gets authorization tokens for the current user or a specific user when using the `-u` flag



Handling secure authorization

Security needs to addressed when moving the Accumulo solution from the development stage to the production stage. Using secure authorization is good practice. For example, public-key infrastructure PKI (read more about it at `http://en.wikipedia.org/wiki/Public-key_infrastructure`), can be used for authentication.

Query Services Layer

Java API is the primary method of interaction with Accumulo. When moving your code from the development to the production environment, there might be a good reason to use Query Services Layer. Query Services Layer provides a platform where the Presentation Layer (web applications), is hosted on Apache Tomcat or Internet Information Service (IIS). At this level, security is often implemented. The normal setup is to have Query Services Layer between Accumulo and the user machine.

Summary

In this chapter, we examined how Accumulo uses cell-level security to have an active and complete control over the visibility of every cell for every table.

Currently, Accumulo is the only NoSQL database that is designed with cell-level security. Cell-level security is very important and has lots of benefits for large organizations. To be able to secure information sharing, and secure information multi-tenancy, you can create a bigger data store without users to see what they don't have access to.

In the appendices, there are the following lists of commands:

- Commands for the Accumulo shell interface
- Commands for Hadoop
- The ZooKeeper commands

Using commands for Accumulo is a great time saver; very similar to having an interface on top of relational databases. Testing everything in the command-line interface before writing the Java code is good practice. For ZooKeeper, the command-line interfaces are more about monitoring than actually doing any work. And finally, the Hadoop command line is for both the user and the administrator with a focus on both managing and monitoring.

A

Accumulo Command References

The commands for the Accumulo shell interface are as follows:

Command	Description
`?`	Lists all available commands.
`about`	Provides information about the Accumulo shell command interface.
`addsplits`	Adds split points to an existing table.
`authenticate`	Verifies a user's credentials.
`bye`	Exits the shell.
`classpath`	Lists the files currently on the classpath.
`clear`	Clears the screen.
`clonetable`	Clones a table.
`cls`	Clears the screen.
`compact`	Sets all tablets for a table to major compact as soon as possible (based on current time).
`config`	Prints system properties and table-specific properties.
`createtable`	Creates a new table.
`createuser`	Creates a new user.
`debug`	Turns debug logging on or off.
`delete`	Deletes a record from a table.
`deleteiter`	Deletes a table-specific iterator.
`deletemany`	Scans a table and deletes the resulting records.
`deleterows`	Deletes a range of rows in a table. Note that rows matching the start row *are not* deleted, but rows matching the end row *are*.

Command	Description
`deletescaniter`	Deletes a table-specific scan iterator so that it is no longer used during this shell session.
`deletetable`	Deletes a table.
`deleteuser`	Deletes a user.
`droptable`	Deletes a table.
`dropuser`	Deletes a user.
`du`	Prints how much space is used by files referenced by a table. When multiple tables are specified, it prints how much space is used by files shared between tables, if any.
`egrep`	Searches each row, column family, column qualifier, and value, in parallel, on the server side (using a Java Matcher, so put `.*` before and after your term if you're not matching the whole element).
`execfile`	Specifies a file containing Accumulo commands to execute.
`exit`	Exits the shell.
`flush`	Flushes to disk the table data that is currently in memory.
`formatter`	Specifies a formatter to use for displaying table entries.
`getauths`	Displays the maximum scan authorizations for a user.
`getgroups`	Gets the locality groups for a given table.
`getsplits`	Retrieves the current split points for tablets in the current table.
`grant`	Grants system or table permissions to a user.
`grep`	Searches each row, column family, column qualifier, and value in a table for a substring (not a regular expression), in parallel, on the server side.
`help`	Provides information about the available commands.
`history`	Generates a list of all commands previously executed.
`importdirectory`	Imports in bulk an entire directory of data files into the current table. The Boolean argument determines whether Accumulo sets the time.
`info`	Displays information about this program.
`insert`	Inserts a record.
`listiter`	Lists table-specific iterators.
`listscans`	Lists what scans are currently running in Accumulo. See the `accumulo.core.client.admin.ActiveScan` Javadoc for more information about columns.
`masterstate`	This has been deprecated. Use the command-line utility instead.

Command	Description
`maxrow`	Finds the row with the maximum value in a table within a given range.
`merge`	Merges tablets in a table.
`notable`	Returns to a table-less shell state.
`offline`	Starts the process of taking the table offline.
`online`	Starts the process of putting the table online.
`passwd`	Changes a user's password.
`quit`	Exits the shell.
`renametable`	Renames a table.
`revoke`	Revokes system or table permissions from a user.
`scan`	Scans the table and displays the resulting records.
`select`	Scans for and displays a single entry.
`selectrow`	Scans a single row and displays all resulting records.
`setauths`	Sets the maximum scan authorizations for a user.
`setgroups`	Sets the locality groups for a given table (for binary or commas, use Java API).
`setiter`	Sets a table-specific iterator.
`setscaniter`	Sets a table-specific scan iterator for this shell session.
`sleep`	Sleeps for the given number of seconds.
`systempermissions`	Displays a list of valid system permissions.
`table`	Switches to the specified table.
`tablepermissions`	Displays a list of valid table permissions.
`tables`	Displays a list of all existing tables.
`trace`	Turns trace logging on or off.
`user`	Switches to the specified user.
`userpermissions`	Displays a user's system and table permissions.
`users`	Displays a list of existing users.
`whoami`	Reports the current username.

B
Hadoop Command References

The commands for Hadoop are as follows:

Command	Description
User commands	
`archive`	Creates a Hadoop archive
`distcp`	Copies files or directories recursively
`fs`	Runs a generic filesystem user client
`fsck`	Runs an HDFS filesystem checking utility
`fetchdt`	Gets the delegation token from a NameNode
`jar`	Runs a JAR file
`job`	Lets you interact with Map/Reduce jobs
`pipes`	Runs a pipes job
`queue`	Lets you interact with and view job queue information
`version`	Prints the version
`classpath`	Prints the class path needed to get the Hadoop JAR files and the required libraries
Administration commands	
`balancer`	Runs a cluster balancing utility
`daemonlog`	Lets you get/set the log level for each daemon
`datanode`	Runs an HDFS `datanode`
`dfsadmin`	Runs an HDFS `dfsadmin` client
`mradmin`	Runs an MR admin client

Command	Description
`jobtracker`	Runs the Map/Reduce JobTracker node
`namenode`	Runs the NameNode
`secondarynamenode`	Runs the HDFS secondary NameNode
`tasktracker`	Runs a Map/Reduce TaskTracker node

C
ZooKeeper Command References

ZooKeeper commands; the four-letter words. You can use the commands via telnet or NC at the client port; for example, like this:

```
echo conf | nc 127.0.0.1 511
```

The following table explains the ZooKeeper commands:

Command	Description
`conf`	Prints details about serving configuration
`cons`	Lists full connection/session details for all clients connected to this server
`crst`	Resets connection/session statistics for all connections
`dump`	Lists the outstanding sessions and ephemeral nodes
`envi`	Prints details about serving environment
`ruok`	Tests if server is running in a non-error state
`srst`	Resets server statistics
`srvr`	Lists full details for the server
`stat`	Lists brief details for the server and connected clients
`wchs`	Lists brief information on watches for the server
`wchc`	Lists detailed information on watches for the server by session
`wchp`	Lists detailed information on watches for the server by path
`mntr`	Outputs a list of variables that could be used for monitoring the health of the cluster

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Thank you for buying

Apache Accumulo for Developers

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

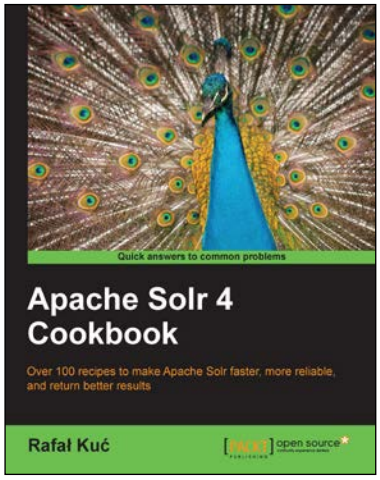
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
community experience distilled
PUBLISHING

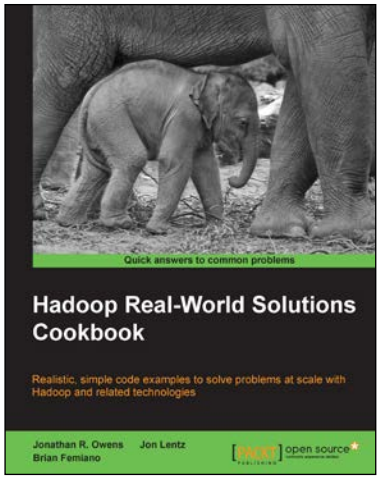


Apache Solr 4 Cookbook

ISBN: 978-1-782161-32-5 Paperback: 328 pages

Over 100 recipes to make Apache Solr faster, more reliable, and return better results

1. Learn how to make Apache Solr search faster, more complete, and comprehensively scalable
2. Solve performance, setup, configuration, analysis, and query problems in no time
3. Get to grips with, and master, the new exciting features of Apache Solr 4



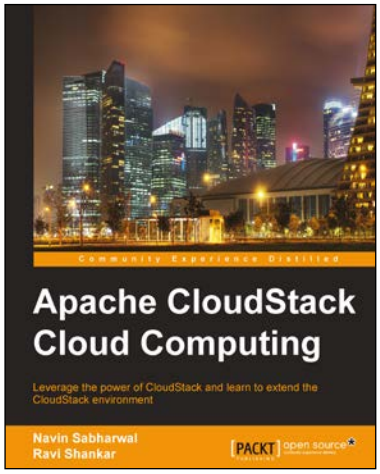
Hadoop Real-World Solutions Cookbook

ISBN: 978-1-849519-12-0 Paperback: 316 pages

Realistic, simple code examples to solve problems at scale with Hadoop and related technologies

1. Solutions to common problems when working in the Hadoop environment
2. Recipes for (un)loading data, analytics, and troubleshooting
3. In depth code examples demonstrating various analytic models, analytic solutions, and common best practices

[PACKT] open source
community experience distilled
PUBLISHING

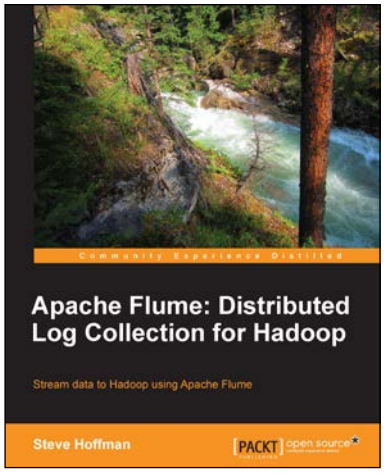


Apache CloudStack Cloud Computing

ISBN: 978-1-782160-10-6 Paperback: 294 pages

Leverage the power of CloudStack and learn to extend the CloudStack environment

1. Install, deploy, and manage a cloud service using CloudStack
2. Step-by-step instructions on setting up and running the leading open source cloud platform CloudStack
3. Set up an IaaS cloud environment using CloudStack



Apache Flume: Distributed Log Collection for Hadoop

ISBN: 978-1-782167-91-4 Paperback: 108 pages

Stream data to Hadoop using Apache Flume

1. Integrate Flume with your data sources
2. Transcode your data en-route in Flume
3. Route and separate your data using regular expression matching