



HDInsight

Succinctly

by James Beresford

HDInsight Succinctly

By

James Beresford

Foreword by Daniel Jebaraj



Copyright © 2014 by Syncfusion, Inc.
2501 Aerial Center Parkway
Suite 200
Morrisville, NC 27560
USA
All rights reserved.

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com upon completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages or any other liability arising from, out of or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: Buddy James

Copy Editor: Suzanne Kattau

Acquisitions Coordinator: Hillary Bowling, marketing coordinator, Syncfusion, Inc.

Proofreader: Darren West, content producer, Syncfusion, Inc.

Table of Contents

Table of Figures	6
The Story behind the <i>Succinctly</i> Series of Books	7
About the Author	9
Aims of this Book.....	10
Chapter 1 Platform Overview.....	11
Microsoft's Big Data Platforms.....	11
Data Management and Storage.....	12
HDInsight and Hadoop.....	12
Chapter 2 Sentiment Analysis	14
A Simple Overview.....	14
Complexities	16
Chapter 3 Using the HDInsight Platform on Azure to Perform Simple Sentiment Analysis	17
Chapter 4 Configuring an HDInsight Cluster	18
Chapter 5 HDInsight and the Windows Azure Storage Blob	20
Loading Data into Azure Blob Storage	20
Referencing Data in Azure Blob Storage.....	21
Chapter 6 HDInsight and PowerShell.....	24
Chapter 7 Using C# Streaming to Build a Mapper	25
Streaming Overview.....	26
Streaming with C#.....	26
Data Source	26
Data Challenges	27
Data Spanning Multiple Lines.....	27
Inconsistent Formatting	29

Quoted Text	30
Words of No Value.....	31
Executing the Mapper against the Data Sample	32
Chapter 8 Using Pig to Process and Enrich Data	35
Using Pig.....	35
Referencing the Processed Data in a Relation.....	36
Joining the Data	38
Aggregating the Data	39
Exporting the Results.....	40
Additional Analysis on Word Counts.....	41
Chapter 9 Using Hive to Store the Output	43
Creating an External Table to Reference the Pig Output	43
Chapter 10 Using the Microsoft BI Suite to Visualize Results	45
The Hive ODBC Driver and PowerPivot	45
Installing the Hive ODBC Driver	45
Setting up a DSN for Hive.....	45
Importing Data into Excel.....	47
Adding Context in PowerPivot	49
Importing a Date Table from Windows Azure DataMarket	50
Creating a Date Hierarchy	51
Linking to the Sentiment Data	53
Adding Measures for Analysis	53
Visualizing in PowerView	55
PowerQuery and HDInsight	59
Other Components of HDInsight	60
Oozie.....	60
Sqoop.....	60
Ambari.....	60

Table of Figures

Figure 1: HDInsight from the Azure portal	18
Figure 2: Creating an HDInsight cluster.....	19
Figure 3: CloudBerry Explorer connected to Azure Storage.....	21
Figure 4: The Hadoop Command Line shortcut.....	35
Figure 5: Invoking the Pig Command Shell.....	36
Figure 6: DUMP output from Pig Command Shell	37
Figure 7: Pig command launching MapReduce jobs	41
Figure 8: ODBC apps.....	46
Figure 9: Creating a new System DSN using the Hive ODBC driver.....	46
Figure 10: Configuring the Hive DSN.....	47
Figure 11: The Excel PowerPivot Ribbon tab	47
Figure 12: Excel PowerPivot Manage Data Model Ribbon	48
Figure 13: Excel PowerPivot Table Import Wizard - Data Source Type selection.....	48
Figure 14: Excel PowerPivot Table Import Wizard - Data Link Type selection	48
Figure 15: Excel PowerPivot Table Import Wizard - Selecting Hive tables	49
Figure 16: Excel PowerPivot Data Model Diagram View	49
Figure 17: Excel PowerPivot Import Data from Data Service	50
Figure 18: Excel Windows Azure Marketplace browser	50
Figure 19: Excel Windows Azure Marketplace data feed options.....	51
Figure 20: Excel PowerPivot Data Model - Creating a hierarchy.....	52
Figure 21: Excel PowerPivot Data Model - Adding levels to a hierarchy.....	52
Figure 22: Adding a measure to the Data Model	54
Figure 23: Launching PowerView in Excel.....	55
Figure 24: PowerView fields browsing	56
Figure 25: PowerView sample report "Author name distribution"	57
Figure 26: PowerView sample report "Sentiment by Post Length"	58
Figure 27: PowerView sample report "Sentiment by Author over Time"	58

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!



About the Author

James Beresford is a certified Microsoft Business Intelligence (BI) Consultant who has been working with the platform for over a decade. He has worked with all aspects of the stack, his specialty being extraction, transformation, and load (ETL) with SQL Server Integration Services (SSIS) and Data Warehousing on SQL Server. He has presented twice at TechEd in Australia and is a frequent presenter at various user groups.

His client experience includes companies in the insurance, education, logistics and banking fields. He first used the HDInsight platform in its preview stage for a telecommunications company to analyse unstructured data, and has watched the platform grow and mature since its early days.

He blogs at www.bimonkey.com and tweets [@BI_Monkey](https://twitter.com/BI_Monkey). He can be found on LinkedIn at <http://www.linkedin.com/in/jamesberesford>.

Aims of this Book

HDInsight Succinctly aims to introduce the reader to some of the core concepts of the HDInsight platform and explain how to use some of the tools it makes available to process data. This will be demonstrated by carrying out a simple Sentiment Analysis process against a large volume of unstructured text data.

This book has been written from the perspective of an experienced BI professional and, consequently, part of this book's focus is on translating Hadoop concepts in those terms as well as on translating Hadoop tools to more familiar languages such as Structured Query Language (SQL) and MultiDimensional eXpressions (MDX). Experience in either of these languages is not required to understand this book but, for those with roots in the relational data world, experience in these languages will help in understanding its content.

Throughout the course of this book, the following features will be demonstrated:

- Setting up and managing HDInsight clusters on Azure
- The use of Azure Blob Storage to store input and output data
- Understanding the role of PowerShell in managing clusters and executing jobs
- Running MapReduce jobs written in C# on the HDInsight platform
- The higher-level languages Pig and Hive
- Connecting with Microsoft BI tools to retrieve, enrich, and visualize the output

The example process will not cover all the features available in HDInsight. In a closing chapter, the book will review some of the features not previously discussed so the reader will have a complete view of the platform.

It is worth noting that the approaches used in this book are not designed to be optimal for performance or process time, as the aim is to demonstrate the capabilities of the range of tools available rather than focus on the most efficient way to perform a specific task. Performance considerations are significant as they will impact not just how long a job takes to run but also its cost. A long-running job consumes more CPU and one that generates a large volume of data—even as temporary files—will consume more storage. When this is paid for as part of a cloud service, the costs can soon mount up.

Chapter 1 Platform Overview

Microsoft's Big Data Platforms

The world of data is changing in a big way and expectations about how to interact and analyze that data are changing as a result. Microsoft offers a broad and scalable portfolio of data storage capabilities for structured, unstructured, and streaming data—both on-premises and in the cloud.

Microsoft has been present in the traditional BI space through the SQL Server platform which scales quite satisfactorily into the hundreds of gigabytes range without too much need for specialist hardware or clever configuration. Since approximately 2010, Microsoft has also offered a couple of specialist appliances to scale higher: the SQL Server Fast Track Data Warehouse for anything up to 100 terabytes, and the SQL Server Parallel Data Warehouse (PDW) for anything entering the petabyte scale.

However, these platforms only deal with relational data and the open-source movement overtook Microsoft (and indeed many other vendors) with the emergence of Hadoop. Microsoft did have a similar platform internally called Dryad but, shortly before Dryad was expected to go live, it was dropped in favor of creating a distribution of Hadoop in conjunction with Hortonworks.^{1 2}

From that decision point, various previews of the platform were made available as on-premises or cloud versions. Early in 2013, the HDInsight name was adopted for the preview (replacing the original “Hadoop on Azure” name) and the cloud platform became generally available in October 2013. The on-premises version is, at the time of this writing, still in preview with no firm release date.

Aspects of these technologies are working their way back into the relational world: The 2.0 version of the Parallel Data Warehouse features support for Hadoop including a language called PolyBase that allows queries to include relational and nonrelational data in the same statements.³

¹ Dryad Project page: <http://research.microsoft.com/en-us/projects/dryad/>

² ZDNet - Microsoft drops Dryad; puts its big-data bets on Hadoop: <http://www.zdnet.com/blog/microsoft/microsoft-drops-dryad-puts-its-big-data-bets-on-hadoop/11226>

³ PolyBase - <http://www.microsoft.com/en-us/sqlserver/solutions-technologies/data-warehousing/polybase.aspx>

Data Management and Storage

Data management needs have evolved from traditional relational storage to both relational and nonrelational storage, and a full-spectrum information management platform needs to support all types of data. To deliver insight on any data, a platform is needed that provides a complete set of capabilities for data management across relational, nonrelational and streaming data. The platform needs to be able to seamlessly move data from one type to another, and be able to monitor and manage all data regardless of the type of data or data structure it is. This has to occur without the application having to worry about scale, performance, security, and availability.

In addition to supporting all types of data, moving data to and from a nonrelational store (such as Hadoop) and a relational data warehouse is one of the key Big Data customer usage patterns. To support this common usage pattern, Microsoft provides connectors for high-speed data movement between data stored in Hadoop and existing SQL Server Data Warehousing environments, including SQL Server Parallel Data Warehouse.

There is a lot of debate in the market today over relational vs. nonrelational technologies. Asking the question, “Should I use relational or nonrelational technologies for my application requirements?” is asking the wrong question. Both are storage mechanisms designed to meet very different needs and the two should be considered as complementary.

Relational stores are good for structured data where the schema is known, which makes programming against a relational store require an understanding of declarative query languages like SQL. These platforms deliver a store with high consistency and transaction isolation.

In contrast, nonrelational stores are good for unstructured data where schema does not exist or where applying it is expensive and querying it is more programmatic. This platform gives greater flexibility and scalability—with a tradeoff of losing the ability to easily work with the data in an ACID manner; however, this is not the case for all NoSQL databases (for example, RavenDB).

As the requirements for both of these types of stores evolve, the key point to remember is that a modern data platform must support both types of data equally well, provide unified monitoring and management of data across both, and be able to easily move and transform data across all types of stores.

HDInsight and Hadoop

Microsoft’s Hadoop distribution is intended to bring the robustness, manageability, and simplicity of Windows to the Hadoop environment.

For the on-premises version, that means a focus on hardening security through integration with Active Directory, simplifying manageability through integration with System Center, and dramatically reducing time to set up and deploy via simplified packaging and configuration. These improvements will enable IT to apply consistent security policies across Hadoop clusters and manage them from a single pane of glass on System Center.

For the service on Windows Azure, Microsoft will further lower the barrier to deployment by enabling the seamless setup and configuration of Hadoop clusters through easy to use components of the Azure management portal.

Finally, they are not only shipping an open source-based distribution of Hadoop but are also committed to giving back those updates to the Hadoop community. Microsoft is committed to delivering 100-percent compatibility with Apache Hadoop application programming interfaces (APIs) so that applications written for Apache Hadoop should work on Windows.

Working closely with [Hortonworks](http://hortonworks.com/), Microsoft has submitted a formal proposal to contribute the Hadoop-based distribution on Windows Azure and Windows Server as changes to the Apache code base.⁴ In addition, they are also collaborating on additional capabilities such as Hive connectivity, and an innovative JavaScript library developed by Microsoft and Hortonworks to be proposed as contributions to the Apache Software Foundation.

Hortonworks is focused on accelerating the development and adoption of Apache Hadoop. Together with the Apache community, they are making Hadoop more robust and easier to use for enterprises, and more open and extensible for solution providers.

As the preview has passed through, various features have come and gone. An original feature was the Console, a friendly web user interface that allowed job submission, access to Hive, and a JavaScript console that allowed querying of the File system and submission of Pig jobs. This functionality has gone but is expected to migrate into the main Azure Portal at some time (though what this means for the on-premises version is unclear). However, in its place has appeared a fully featured set of PowerShell cmdlets that allows remote submission of jobs and even creation of clusters.

One feature that has remained has been the ability to access Hive directly from Excel through an Open Database Connectivity (ODBC) driver. This has enabled the consumption of the output of Hadoop processes through an interface with which many users are familiar, and connects Hadoop with the data mashup capabilities of PowerPivot and rich visualizations of PowerView.

The platform continues to evolve and features are constantly arriving (and occasionally going). This book will do its best to capture the current state but, even as it was being written, content needed to be updated to deal with the ongoing changes.

⁴ Hortonworks company website: <http://www.hortonworks.com/>

Chapter 2 Sentiment Analysis

To help get a grasp on the tools within HDInsight we will demonstrate their usage through a applying a simple Sentiment Analysis process to a large volume of unstructured text data. In this short non-technical section we will look at what Sentiment Analysis is. As part of this a simple approach will be set down which is the one that will be used as we progress through our exploration of HDInsight.

A Simple Overview

Sentiment Analysis is the process of deriving emotional context from communications through analyzing the words and terms used in those communications. This can be spelled out in the simple example below:

Step 1: Take some simple free-form text such as text from a hotel review:

Title	Hotel Feedback
Content	I had a fantastic time on holiday at your resort. The service was excellent and friendly. My family all really enjoyed themselves. The pool was closed, which kind of sucked though.

Step 2: Take a list of words deemed as “positive” or “negative” in Sentiment:

Positive	Negative
Good	Bad
Great	Worse
Fantastic	Rubbish
Excellent	Sucked
Friendly	Awful
Awesome	Terrible
Enjoyed	Bogus

Step 3: Match the text to the Sentiment word list:

Title	Hotel Feedback
Content	<p>I had a fantastic time on holiday at your resort. The service was excellent and friendly. My family all really enjoyed themselves.</p> <p>The pool was closed, which kind of sucked though.</p>

Step 4: Count the Sentiment words in each category:

Positive	Negative
Fantastic	Sucked
Excellent	
Friendly	
Enjoyed	
4	1

Step 5: Subtract the negative from the positive:

Positive Sentiment	4
Negative Sentiment	1
Overall Sentiment	3

In this example, the overall result is that the Sentiment of this particular block of text is positive and an automated system could interpret this as a positive review.

Complexities

The view presented above is a very simplistic approach to Sentiment Analysis, as it examines individual words free of context and decides whether they are positive or negative. For example, consider this paragraph:

"I think you misunderstand me. I do not hate this and it doesn't make me angry or upset in any way. I just had a terrible journey to work and am feeling a bit sick."

By examining it using human ability to derive context, this is not a negative comment at all; it is quite apologetic. But it is littered with words that, assessed in isolation, would present a view that was very negative. Simple context can be added by considering the influence of modifying words such as "not", though this has an impact on processing time. More complex context starts entering the domain of Natural Language Processing (NLP) which is a deep and complicated field that attempts to address these challenges.

A second issue is in the weight that is given to particular words. "Hate" is a stronger expression of dislike than "dislike" is—but where on that spectrum are "loathe" and "sucks"? A given person's writing style would also impact the weight of such words. Someone prone to more dramatic expressions may declare that they "hate" something that is just a minor inconvenience, when a more diplomatic person may state that they are "concerned" about something that actually has caused them great difficulty.

This can be addressed in a couple of ways. The first way is to set aside the individual's style and apply weighting to specific words according to a subjective judgment. This, of course, presents the challenge that the list of words will be long and, therefore, assigning weights will be a time-consuming effort. Also, it is quite probable that not all the words will be encountered in the wild. The second way—and one that reflects a technique used in the analytical world when addressing outcomes on a scale that is not absolute—is to simply use a simplistic approach that allocates a word as positive, negative or, in the absence of a categorization, neutral—and set the scale issue to one side.

A third issue is the distribution and use of words in a given scenario. In some cases, words that are common in the domain being analyzed may give false positives or negatives. For example, a pump manufacturer looking at reviews of its products should not be accounting for the use of the word "sucks" as it is a word that would feature in descriptions of those products' capabilities. This is a simpler issue to address as, like part of any Sentiment Analysis, it is important to review the more frequent words that are impacting Sentiment in case words are being assessed as doing so when they are actually neutral in that specific domain.

For further reading on this field, it is recommended you look at the work of University of Illinois professor Bing Liu (an expert in this field) at <http://www.cs.uic.edu/~liub/>.

Chapter 3 Using the HDInsight Platform on Azure to Perform Simple Sentiment Analysis

In this book, we will be discussing how to perform a simple, word-based Sentiment Analysis exercise using the HDInsight platform on Windows Azure. This process will consist of several steps:

- Creating and configuring an HDInsight cluster
- Uploading the data to Azure Blob Storage
- Creating a Mapper to decompose the individual words in a message using C# streaming
- Executing that Mapper as a Hadoop MapReduce job
- Using Pig to:
 - apply Sentiment indicators to each word within a message
 - aggregate the Sentiment across messages and words
 - exporting the aggregated results back to Azure Blob Storage
- Using Hive to expose the results to ODBC
- Adding context using PowerPivot
- Visualizing using PowerView

Chapter 4 Configuring an HDInsight Cluster

Configuring an HDInsight cluster is designed to be an exercise that demonstrates the true capacity of the cloud to deliver infrastructure simply and quickly. The process of provisioning a nine-node cluster (one head node and eight worker nodes) can take as little as 15 minutes to complete.

HDInsight is delivered as part of the range of services available through the Windows Azure platform. HDInsight was formally launched as a publicly available service in October 2013.

Once access to the program is granted, HDInsight appears in the selection of available services:

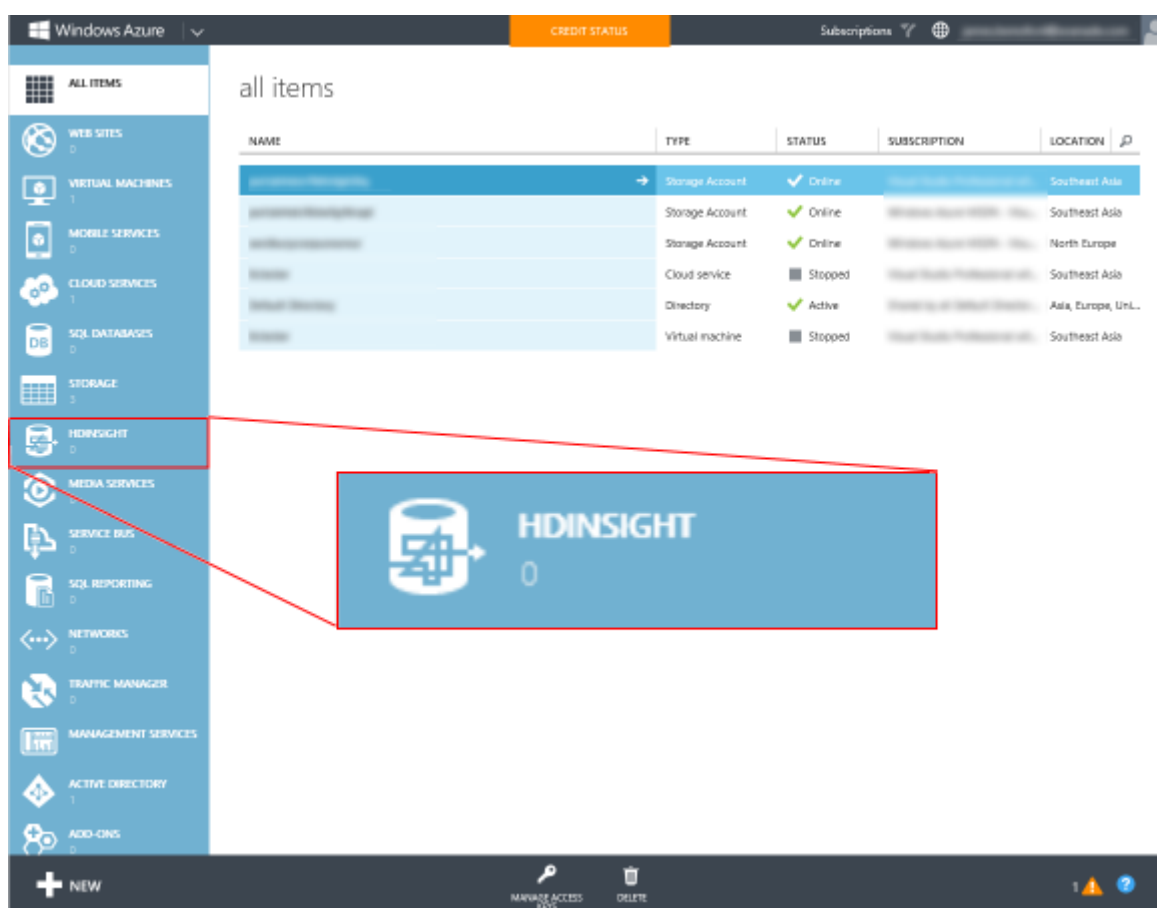


Figure 1: HDInsight from the Azure portal

To create a cluster, select the HDInsight Service option and you will be directed to create one. To do so, you will be directed to the Quick Create option which will create a cluster using some basic presets. Cluster sizes are available from four nodes to 32 nodes. You will need an Azure storage account in the same region as your HDInsight cluster to hold your data. This will be discussed in a later section.

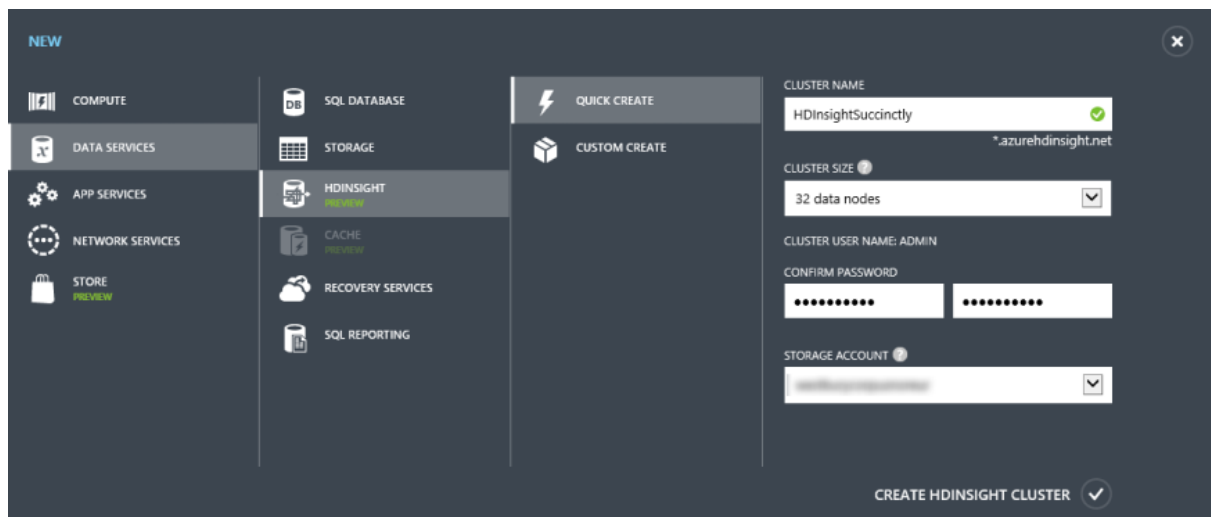


Figure 2: Creating an HDInsight cluster

While you may be tempted to create the biggest cluster possible, a 32-node cluster could cost US\$261.12 per day to run and may not necessarily give you a performance boost depending on how your job is configured.⁵

If you opt to custom create, you gain flexibility over selecting your HDInsight version, exact number of nodes, location, ability to select Azure SQL for a Hive and Oozie metastore, and finally, more options over storage accounts including selecting multiple accounts.

⁵ As per pricing quoted at time of writing from: <http://www.windowsazure.com/en-us/pricing/details/hdinsight/>

Chapter 5 HDInsight and the Windows Azure Storage Blob

Loading Data into Azure Blob Storage

The HDInsight implementation of Hadoop can reference the Windows Azure Storage Blob (WASB) which provides a full-featured Hadoop Distributed File System (HDFS) over Azure Blob Storage.⁶ This separates the data from compute nodes. This conflicts with the general Hadoop principle of moving the data to the compute in order to reduce network traffic, which is often a performance bottleneck. This bottleneck is avoided in WASB as it streams data from Azure Blob Storage over the fast Azure Flat Network Storage—otherwise known as the “Quantum 10” (Q10) network architecture—which ensures high performance.⁷

This allows you to store data on cheap Azure Storage rather than maintaining it on the significantly more expensive HDInsight cluster’s compute nodes’ storage. It further allows for the relatively slow process of uploading data to precede launching your cluster and allows your output to persist after shutting down the cluster. This makes the compute component genuinely transitional and separates the costs associated with compute from those associated with storage.

Any Hadoop process can then reference data on WASB and, by default, HDInsight uses it for all storage including temporary files. The ability to use WASB applies to not just base Hadoop functions but extends to higher-level languages such as Pig and Hive.

Loading data into Azure Blob Storage can be carried out by a number of tools. Some of these are listed below:

Name	GUI	Free	Source
AzCopy	No	Yes	http://blogs.msdn.com/b/windowsazurestorage/archive/2012/12/03/azcopy-uploading-downloading-files-for-windows-azure-blobs.aspx
Azure Storage Explorer	Yes	Yes	http://azurestorageexplorer.codeplex.com/
CloudBerry Explorer for Azure Storage	Yes	Yes	http://www.cloudberrylab.com/free-microsoft-azure-explorer.aspx

⁶ Azure Vault Storage in HDInsight: A Robust and Low Cost Storage Solution: <http://blogs.msdn.com/b/silverlining/archive/2013/01/29/azure-vault-storage-in-hdinsight-a-robust-and-low-cost-storage-solution.aspx>

⁷ Why use Blob Storage with HDInsight on Azure: <http://dennyglee.com/2013/03/18/why-use-blob-storage-with-hdinsight-on-azure/>

Name	GUI	Free	Source
CloudXplorer	Yes	No	http://clumsyleaf.com/products/cloudexplorer
Windows and SQL Azure tools for .NET professionals	Yes	No	http://www.red-gate.com/products/azure-development/

A screenshot of CloudBerry Explorer connected to the Azure Storage being used in this example is below:

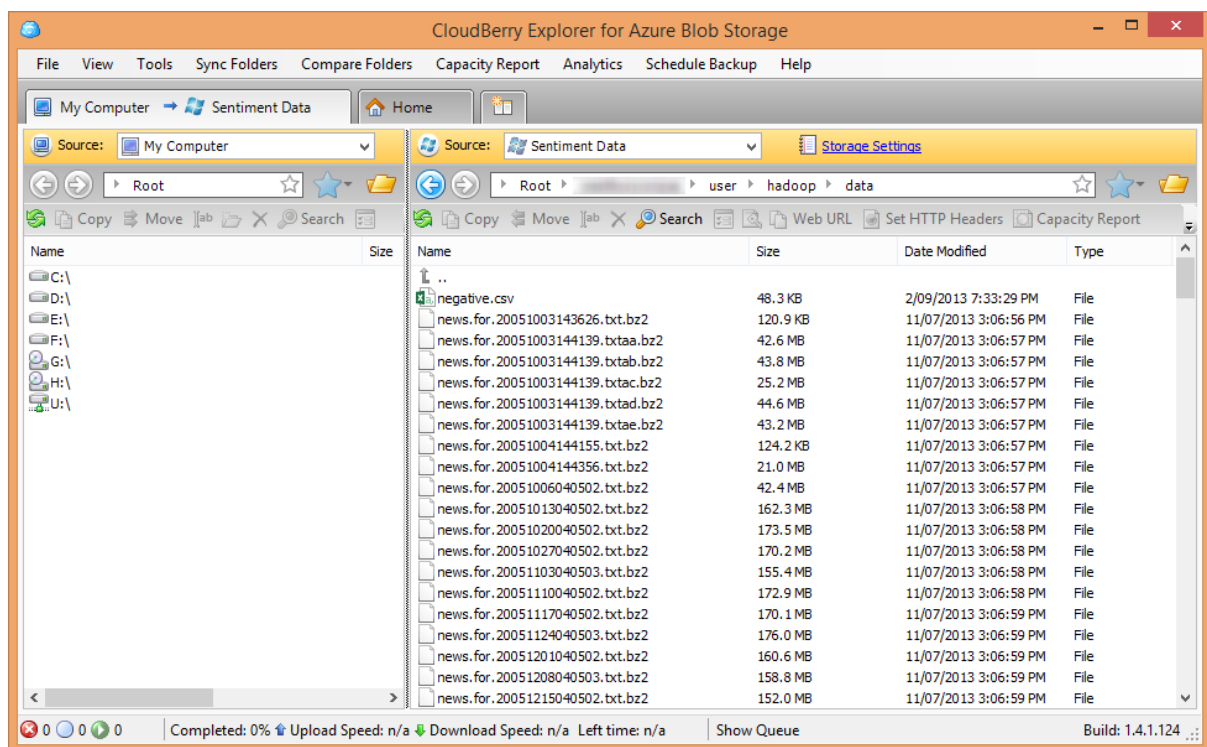


Figure 3: CloudBerry Explorer connected to Azure Storage

As you can see, it is presented very much like a file explorer and most of the functionality you would expect from such a utility is available.

Uploading significant volumes of data for processing can be a time-consuming process depending on available bandwidth, so it is recommended that you upload your data before you set up your cluster as these tasks can be performed independently. This stops you from paying for compute time while you wait for data to become available for processing.

Referencing Data in Azure Blob Storage

The approach to referencing data held in the WASB depends on the configuration of the HDInsight instance.

When creating the HDInsight cluster in the Management Portal using the Quick Create option, you specify an existing storage account. Creating the cluster will also cause a new container to be created in that account. Using Custom Create, you can specify the container within the storage account.

Normal Hadoop file references look like this:

```
hdfs://[name node path]/directory level 1/directory level 2/filename
```

eg:

```
hdfs://localhost/user/data/big_data.txt
```

WASB references are similar except, rather than referencing the name node path, the Azure Storage container needs to be referenced:

```
wasb[s]://[<container>@]<accountname>.blob.core.windows.net/<path>
```

eg:

```
wasb://datacontainer@storageaccount.blob.core.windows.net/user/data/big_data.txt
```

For the default container, the explicit account/container information can be dropped, for example:

```
hadoop fs -ls wasb://user/data/big_data.txt
```

It is even possible to drop the wasb:// reference as well:

```
hadoop fs -ls user/data/big_data.txt
```

Note the following options in the full reference:

- * wasb[s]: the [s] allows for secure connections over SSL
- * The container is optional for the default container

The second point is highlighted because it is possible to have a number of storage accounts associated with each cluster. If using the Custom Create option, you can specify up to seven additional storage accounts.

If you need to add a storage account after cluster creation, the configuration file core-site.xml needs to be updated, adding the storage key for the account so the cluster has permission to read from the account using the following XML snippet:

```
<property>
<name>fs.azure.account.key.[accountname].blob.core.windows.net</name>
>

<value>[accountkey]</value> </property>
```

Complete documentation can be found on the Windows Azure website.⁸

As a final note, the wasb:// notation is used in the higher-level languages (for example, Hive and Pig) in exactly the same way as it is for base Hadoop functions.

⁸ Using Windows Azure Blob Storage with HDInsight: <http://www.windowsazure.com/en-us/manage/services/hdinsight/howto-blob-store/>

Chapter 6 HDInsight and PowerShell

PowerShell is the Windows scripting language that enables manipulation and automation of Windows environments.⁹ It is an extremely powerful utility that allows for execution of tasks from clearing local event logs to deploying HDInsight clusters on Azure.

When HDInsight went into general availability, there was a strong emphasis on enabling submission of jobs of all types through PowerShell. One motivation behind this was to avoid some of the security risks associated with having Remote Desktop access to the head node (a feature now disabled by default when a cluster is built, though easily enabled through the portal). A second driver was to enable remote, automated execution of jobs and tasks. This gives great flexibility in allowing efficient use of resources. Say, for example, web logs from an Azure-hosted site are stored in Azure Blob Storage and, once a day, a job needs to be run to process that data. Using PowerShell from the client side, it would be possible to spin up a cluster, execute any MapReduce, Pig or Hive jobs, and store the output somewhere more permanent such as a SQL Azure database—and then shut the cluster back down.

To cover PowerShell would take a book in itself, so here we will carry out a simple overview. More details can be found on TechNet.¹⁰

PowerShell's functionality is issued through cmdlets. These are commands that accept parameters to execute certain functionality.

For example, the following cmdlet lists the HDInsight clusters available in the specified subscription in the console:

```
Get-AzureHDInsightCluster -Subscription $subid
```

For job execution, such as committing a Hive job, cmdlets look like this:

```
Invoke-Hive "select * from hivesampletable limit 10"
```

These act in a very similar manner to submitting jobs directly via the command line on the server.

Full documentation of the available cmdlets is available on the Hadoop (software development kit (SDK) page on CodePlex.¹¹

Installing the PowerShell extensions is a simple matter of installing a couple of packages and following a few configuration steps. These are captured in the official documentation.¹²

⁹ Scripting with Windows PowerShell: <http://technet.microsoft.com/en-us/library/bb978526.aspx>

¹⁰ Windows PowerShell overview: <http://technet.microsoft.com/en-us/library/cc732114%28v=ws.10%29.aspx>

¹¹ Microsoft .NET SDK for Hadoop: <https://hadoop-sdk.codeplex.com/>

¹² Install and configure PowerShell for HDInsight: <http://azure.microsoft.com/en-us/documentation/services/hdinsight/>

Chapter 7 Using C# Streaming to Build a Mapper

A key component of Hadoop is the MapReduce framework for processing data. The concept is that execution of the code that processes the data is sent to the compute nodes, which is what makes it an example of distributed computing. This work is split across a number of jobs that perform specific tasks.

The Mappers' job is equivalent to the extract components of the ETL paradigm. They read the core data and extract key information from it, in effect imposing structure on the unstructured data. As an aside, the term "unstructured" is a bit of a misnomer in that the data is not without structure altogether—otherwise it would be nearly impossible to parse. Rather, the data does not have structure formally applied to it as it would in a relational database. A pipe delimited text file could be considered unstructured in that sense. So, for example, our source data may look like this:

```
1995|Johns, Barry|The Long Road to  
Succintness|25879|Technical  
  
1987|Smith, Bob|I fought the data and the data  
won|98756|Humour  
  
1997|Johns, Barry|I said too little last  
time|105796|Fictions
```

A human eye may be able to guess that this data is perhaps a library catalogue and what each field is. However, a computer would have no such luck as it has not been told the structure of the data. This is, to some extent, the job of the Mapper. It may be told that the file is pipe delimited and it is to extract the Author's Name as a Key and the Number of Words as the Value as a <Key,Value> pair. So, the output from this Mapper would look like this:

```
[key] <Johns, Barry> [value] <25879>  
[key] <Smith, Bob> [value] <98756>  
[key] <Johns, Barry> [value] <105796>
```

The Reducer is equivalent to the transform component of the ETL paradigm. Its job is to process the data provided. This could be something as complex as a clustering algorithm or something as simple as aggregation (for instance, in our example, summing the Value by the Key), for example:

```
[key] <Johns, Barry> [value] <131675>  
[key] <Smith, Bob> [value] <98756>
```

There are other components to this process, notably Combiners which perform some functions of the Reducer task on an individual Mapper's node. There are also Partitioners which designate where Reducer output gets sent for Reducers to process. For full details, refer to the official documentation at <http://wiki.apache.org/hadoop/HadoopMapReduce>.

It is possible to write some jobs in .NET languages and we will explore this later.

Streaming Overview

Streaming is a core part of Hadoop functionality that allows for the processing of files within HDFS on a line-by-line basis.¹³ The processing is allocated to a Mapper (and, if required, Reducer) that is coded specifically for the exercise.

The process normally operates with the Mapper reading a file chunk on a line-by-line basis, taking the input data from each line (STDIN), processing it, and emitting it as a Key / Value pair to STDOUT. The Key is any data up to the first tab character and the value of whatever follows. The Reducer will then consume data from STDOUT, and process and display it as required.

Streaming with C#

One of the key features of streaming is that it allows languages other than Java to be used as the executable that carries out Map and Reduce tasks. C# executables can, therefore, be used as Mappers and Reducers in a streaming job.

Using `Console.ReadLine()` to process the input (from STDIN) and `Console.WriteLine()` to write the output (to STDOUT), it is easy to implement C# programs to handle the streams of data.¹⁴

In this example, a C# program was written to handle the preprocessing of the raw data as a Mapper, with further processing handled by higher-level languages such as Pig and Hive.

The code referenced below can be downloaded from <https://bitbucket.org/syncfusiontech/hdinsight-succinctly/downloads> as "Sentiment_v2.zip". A suitable development tool such as Visual Studio will be required to work with the code.

Data Source

For this example, the data source was the Westbury Lab Usenet Corpus, a collection of 28 million anonymized Usenet postings from 47,000 groups covering the period between October 2005 and Jan 2011.¹⁵ This is free-format, English text input by humans and presented a sizeable (approximately 35GB) source of data to analyze.

¹³ Hadoop 1.2.1 documentation: <http://hadoop.apache.org/docs/r1.2.1/streaming.html>

¹⁴ An introductory tutorial on this is available on TechNet: <http://social.technet.microsoft.com/wiki/contents/articles/13810.hadoop-on-azure-c-streaming-sample-tutorial.aspx>

¹⁵ Westbury Lab Usenet Corpus: <http://www.psych.ualberta.ca/~westburylab/downloads/usenetcorpus.download.html>

From this data, we could hope to extract the username of the person making the Usenet post, the approximate date and time of the posting, as well as the breakdown of the content of the message.

Data Challenges

There were a number of specific challenges faced when ingesting this data:

- Data for one item spanned across multiple lines
- Format for location of Author name was inconsistent
- The posts frequently contained large volumes of quoted text from other posts
- Some words made up a significant portion of the data without adding insight

Handling these is analyzed in a deeper manner as they are indicative of the type of challenges faced when processing unstructured data.

Data Spanning Multiple Lines

The data provided had a particular challenge for streaming: the text in the data being provided was split across multiple lines. Streaming processes on a line-by-line basis, so it was necessary to retain metadata across multiple lines. An example of this would appear as follows in our data sample:

Data Sample
H.Q.Blanderfleet wrote: I called them and told them what had happened...and asked how if I couldn't get broadband on this new number I'd received the email in the first place ? ---END.OF.DOCUMENT---

This would be read by STDIN as follows:

Line #	Text
1	H.Q.Blanderfleet wrote:
2	

Line #	Text
3	I called them and told them what had happened...and asked how if I couldn't
4	get broadband on this new number I'd received the email in the first place ?
5	
6	---END.OF.DOCUMENT---

This meant that the Mapper had to be able to:

- Identify new elements of data
- Maintain metadata across row reads
- Handle data broken up into blocks on HDFS

Identifying new elements of data was kept simple as each post was delimited by a fixed line of text reading “---END.OF.DOCUMENT---”. In this way, the Mapper could safely assume that finding that text signified the end of the current post.

The second challenge was met by retaining metadata across row reads within normal variables, and resetting them when an end of row was identified. The metadata was emitted attached to each Sentiment keyword.

The third challenge was to address the fact that the data files could be broken up by file chunking on HDFS, meaning that rows would end prematurely in one file and start midblock in another as shown below:

File	Line #	Text
A	1	H.Q.Blanderfleet wrote:
A	2	
A	3	I called them and told them what had happened...and asked how if I couldn't
File Split		
B	4	get broadband on this new number I'd received the email in the first place ?

File	Line #	Text
B	5	
B	6	---END.OF.DOCUMENT---

This was handled in a simple manner. As File A terminated, it would have emitted all of the data it had collected up to that point. File B would simply discard those first rows as invalid as it could not attach metadata to them. This is a compromise that would result in a small loss of data.

Inconsistent Formatting

Within the archive, most messages started with a variable number of blank lines followed by opening lines that sometimes, but not always, indicated the author of the post. This was generally identifiable by the pattern “[Username] wrote:”.

However, this was not consistent as various Usenet clients allowed this to be changed, did not follow a standard format or sometimes the extraction process dropped certain details. Some examples of opening lines are below:

Opening Lines	Comment
"BasketCase" < <EMAILADDRESS> > wrote in message <NEWSURL> ...	As expected, first line holds some text prior to the word “wrote”.
> On Wed, 28 Sep 2005 02:13:52 -0400, East Coast Buttered > < <EMAILADDRESS> > wrote: >	The first line of text does not contain the word “wrote”— it has been pushed to the second line.

Opening Lines	Comment
<p>Once upon a time...long...long ago...I decided to get my home phone number</p> <p>changed...because I was getting lots of silly calls from even sillier</p>	<p>The text does not contain the author details.</p>
<p>On Thu, 29 Sep 2005 13:15:30 +0000 (UTC), "Foobar" wrote:</p>	<p>The author's name had been preceded by a Date/Time stamp.</p>
<p>"Anonnymouse" < <EMAILADDRESS> > proposed that:</p> <p><NEWSURL> ...</p>	<p>The poster had changed from the default word "wrote" to "proposed that".</p>

As an initial compromise, the Mapper simply ignored all the nonstandard cases and marked them as having an "Unknown" author.

In a refinement, regular Expressions were used to match some of the more common date stamp formats and remove them. The details of this are captured in the code sample.

Quoted Text

Within Usenet posts, the default behavior of many clients was to include the prior message as quoted text. For the purposes of analyzing the Sentiment of a given message, this text needed to be excluded as it was from another person.

Fortunately, this quoted text was easily identified as quoted lines began with a “>”, so the Mapper simply discarded any line commencing with this character. This may have resulted in a small but tolerable loss of data (if the author of the post had a line that either intentionally or otherwise started with a “>” character). This was effected as below:

```
// Read line by line from STDIN
while ((line = Console.ReadLine()) != null)
{
    // Remove any quoted posts as identified by starting
    with ">"
    if (line.StartsWith(">") != true)
    {
        //There is no ">" so process data

    }
    ...
}
```

Words of No Value

Some words in the English language are extremely common and would add no insight to a simple, one-word-based Sentiment Analysis.

After an initial pass of the data, it became apparent that there were a large number of two-letter words (“of” and “at”) and single characters (“a” and “i”) which could be ignored, especially given that our Sentiment keyword list had no entries for words less than three characters in length. Consequently, a filter was applied that prevented the display of any string less than three characters in length:

```
// Only write to STDOUT if there is content, ignoring words
of 2 characters or less
if (descword.Length > 2)
{
    Console.WriteLine("{0}", MessageId + "|" + AuthorId +
        "|" + descword);
}
```

Furthermore, there were a few very high frequency words that were of no value such as “and” and “the”. We used a Dictionary and lookup to prevent these from being displayed, modifying the above code to:

```
// Set up some words to ignore
```

```

Dictionary<string, int> IgnoreWords = new Dictionary<string,
int>();

IgnoreWords.Add("the", 1);
IgnoreWords.Add("and", 1);

// Only write to STDOUT if there is content, ignoring words
of 2 characters or less
if (descword.Length > 2)
{
    // Check if in list of ignore words, only write if not
    if (!IgnoreWords.TryGetValue(descword, out value))
    {
        Console.WriteLine(string.Format("{0} | {1} | {2}",
MessageId ,AuthorId, descword));
    }
}
}

```

This significantly reduced the amount of rows to be displayed and, therefore, subsequently processed.

Executing the Mapper against the Data Sample

The raw data provided was in the bzip2 format.¹⁶ Hadoop jobs can process certain compressed file formats natively and also display results in a compressed format if instructed. This means, for executing the sample, no decompression was required in order to process the data. The following formats are confirmed to be supported in HDInsight:

Format	Codec	Extension	Splittable
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec	.deflate	N
gzip	org.apache.hadoop.io.compress.GzipCodec	.gz	N
bzip2	org.apache.hadoop.io.compress.BZip2Codec	.bz2	Y

¹⁶ Wikipedia page on bzip2: <http://en.wikipedia.org/wiki/Bzip2>

The use of compressed input and compressed output has some performance implications which need to be balanced against storage and network traffic considerations. For a full review of these considerations in HDInsight, it is advised you read the white paper by Microsoft on the subject entitled, "Compression in Hadoop" (from which the information in the table above was taken).¹⁷

The Mapper was built as a standard C# console application executable. For the Hadoop job to be able to use it, it needed to be loaded somewhere the job could reference the file. Azure Blob Storage is an obvious and convenient place to handle this.

Once the data and Mapper were loaded, the Hadoop command line was used to specify and launch the job. It is also possible to submit jobs via the SDK or PowerShell.

The full syntax of the job is set out below:

```
c:\apps\dist\hadoop-1.1.0-SNAPSHOT\bin\hadoop.cmd

jar C:\apps\dist\hadoop-1.1.0-SNAPSHOT\lib\hadoop-streaming.jar

"-D mapred.output.compress=true"

"-D
mapred.output.compression.codec=org.apache.hadoop.io.compress
s.GzipCodec"

-files "wasb://user/hadoop/code/Sentiment_v2.exe"

-numReduceTasks 0

-mapper "Sentiment_v2.exe"

-input "wasb://user/hadoop/data"

-output "wasb://user/hadoop/output/Sentiment/"
```

The parameters of the job are explained below¹⁸:

Parameter	Detail
"-D mapred.output.compress=true"	Compress the output
"-D mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec"	Use the GzipCodec to compress the output

¹⁷ Compression in Hadoop: [http://download.microsoft.com/download/1/C/6/1C66D134-1FD5-4493-90BD-98F94A881626/Compression%20in%20Hadoop%20\(Microsoft%20IT%20white%20paper\).docx](http://download.microsoft.com/download/1/C/6/1C66D134-1FD5-4493-90BD-98F94A881626/Compression%20in%20Hadoop%20(Microsoft%20IT%20white%20paper).docx)

¹⁸ For all available parameters, see: <http://hadoop.apache.org/docs/r1.2.1/streaming.html#Generic+Command+Options>

Parameter	Detail
-files "wasb://user/hadoop/code/Sentiment_v2.exe"	Reference the Mapper code in the Azure Blob Storage
-numReduceTasks 0	Specifying there are no Reducer tasks
-mapper "Sentiment_v2.exe"	Specifying the file that is the Mapper
-input "wasb://user/hadoop/data"	Specifying the input directory
-output "wasb://user/hadoop/output/Sentiment/"	Specifying the output directory

A sample of the job results looks like this:

```

276.0|5|bob|government
276.0|5|bob|telling
276.0|5|bob|opposed
276.0|5|bob|liberty
276.0|5|bob|obviously
276.0|5|bob|fail
276.0|5|bob|comprehend
276.0|5|bob|qualifier
276.0|5|bob|legalized
276.0|5|bob|curtis

```

Chapter 8 Using Pig to Process and Enrich Data

As per the official [Apache Pig project page](#):



Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization which, in turn, enables them to handle very large data sets.

In practice, Pig is a language that allows you to describe data sets stored as raw files such as delimited text and, subsequently, perform a series of operations on those data sets that are familiar to SQL developers (such as adding calculations to individual rows or joining and aggregating sets together). It is architected in such a way as to allow the jobs to be massively parallelized using the MapReduce paradigm, as Pig commands are transformed into MapReduce jobs in order to execute. This is not exposed to the Pig programmer directly.

Using Pig

There is no GUI available for Pig at the time of this writing. All commands are executed via a command line on the head node or via PowerShell cmdlets from a client desktop.¹⁹ In this case, we will use the command line which, when using the HDInsight platform, is accessed via the Hadoop command shell (a link to which is on the desktop):

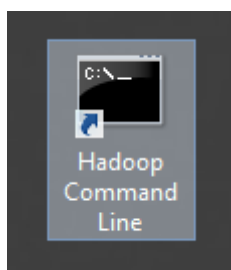


Figure 4: The Hadoop Command Line shortcut

¹⁹ Using Pig with HDInsight: <http://www.windowsazure.com/en-us/manage/services/hdinsight/using-pig-with-hdinsight/>

At the command line, type “pig” and hit enter. This will enter the Pig Command Shell, clearly identifiable as the command prompt changes to “grunt>”:

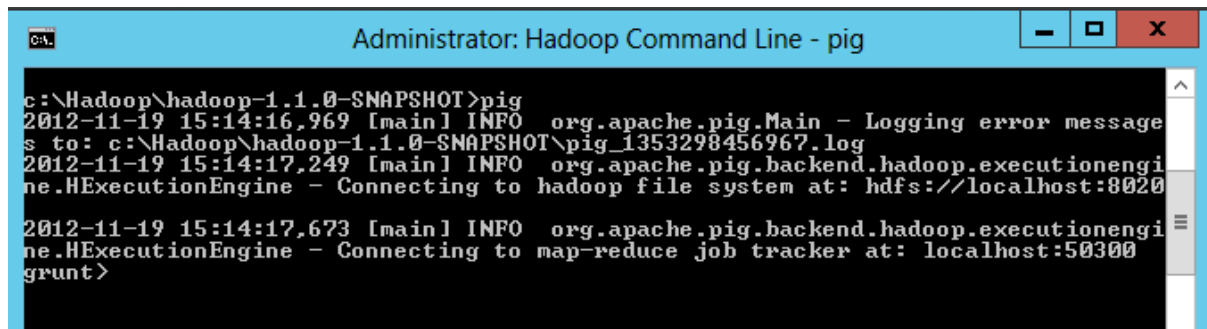


Figure 5: Invoking the Pig Command Shell

From here you can enter Pig commands as described in the documentation.²⁰

Referencing the Processed Data in a Relation

Our first step is to reference the data output by the C# Mapper and the Sentiment keyword lists. Note that I deliberately do not say *load*. At this point, no data is processed and no validation against the source data occurs. Pig only receives a description of the files to be used:

```
data raw = LOAD
'wasb://<container>@<storageaccount>.blob.core.windows.net/u
ser/hadoop/output/Sentiment/part-*' USING PigStorage('|') AS
(filename:chararray,message_id:chararray,author_id:chararray
,word:chararray);
```

Here we use the LOAD command to describe the files we want to reference as the relation “data_raw”, using wildcard characters as supported by Hadoop Globbing.²¹ It is also worth noting that no instructions need to be provided to Pig to tell it that the raw data is compressed. It can handle the compression natively and determines which decompression codec to use based on the data file extension.

As per the documentation, a relation is a bag of tuples which, in turn, is an ordered set of fields. This is a different structural approach to the relational database world, so the callout below explains the concepts (though further reading is recommended):

Relations, bags, tuples, and fields

²⁰ Pig 0.10.0 documentation: <http://pig.apache.org/docs/r0.10.0/>

²¹ Hadoop Globbing documentation <http://hadoop.apache.org/docs/r0.21.0/api/org/apache/hadoop/fs/FileSystem.html#globStatus%28org.apache.hadoop.fs.Path%29>

Starting from the bottom up, a **Field** is very much like a simplified column in the relational database world. It has a name, a data type, and a value, and can be referenced using a zero-based ordinal position within the tuple.

A **Tuple** is similar to a row in a relational database in that it is an ordered collection of fields.

A **Bag** is a collection of tuples. However, this starts deviating from the relational model in that there are two types of bags: the **inner** and **outer**.

Inner Bags are a collection of tuples within tuples, for example:
Tuple 1 ({Tuple 2},{Tuple 3})

Outer Bags are the overall collection of the tuples otherwise known as a **Relation**.

Having referenced the data, we can test our initial data structure to make sure it is sound:

```
temp = LIMIT data_raw 10;  
  
DUMP temp;
```

Here we use the DUMP command to generate some output from Pig and look at how it is interpreting the data. First, we create a relation called “temp” that references our starting relation “data_raw” with a LIMIT command that chooses just the first 10 tuples it finds. Note that this may not be consistent between jobs. Then, by issuing the command to DUMP it, it generates output to the console as below:

```
2012-11-28 15:06:02,532 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!  
2012-11-28 15:06:02,550 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1  
2012-11-28 15:06:02,552 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,iso  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,five  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,least  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,three  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,years  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,minimum  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,requires  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,environment  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,manufacturing  
(hdfs://localhost:8020/user/hadoop/data/news.for.20051003144139.txt.134217728,24  
2e615d-7523-4907-b9e4-a2b324b4390c-1,Unknown,qualifications  
grunt>
```

Figure 6: DUMP output from Pig Command Shell

From this, we can see that our words have a tab appended to the end as shown by the even lining up of the closing brackets around the tuple.

To address this, we use a simple operator to TRIM the last word. We also need to extract the relevant date information from the filename (this could have been done more efficiently in the Mapper but this is purely for demonstrating Pig's capabilities):

```
data_clean = FOREACH data_raw GENERATE
SUBSTRING(filename,48,52) AS year, SUBSTRING(filename,52,54)
AS month, SUBSTRING(filename,54,56) AS day, message id,
author_id, TRIM(word) AS word;
```

The FOREACH operator processes columns of data to GENERATE output. In this case:

- Substrings of the filename to extract Year, Month, and Day values
- A modified value for the “word” field which has white space (which includes tabs) stripped from the start and end of the string

Joining the Data

First, we need to load our Sentiment data word list. The lists used were sourced from work by Bing Liu and Mingqing Hu from the University of Illinois.^{22 23} The lists were loaded directly into Pig with no preprocessing (other than stripping out the file header in a text editor) using the following LOAD command:

```
positive_words = LOAD
'wasb://<container>@<storageaccount>.blob.core.windows.net/u
ser/hadoop/data/positive.csv' USING PigStorage('|') AS
(positive:chararray);

negative_words = LOAD
'wasb://<container>@<storageaccount>.blob.core.windows.net/u
ser/hadoop/data/negative.csv' USING PigStorage('|') AS
(negative:chararray);
```

To add a value to the Sentiment for downstream processing, we add a Sentiment value to each of the lists, assigning a value of 1 to positive words and -1 to negative words using a FOREACH / GENERATE operation:

```
positive = FOREACH positive_words GENERATE positive AS
sentiment_word, 1 AS sentiment_value;

negative = FOREACH negative_words GENERATE negative AS
sentiment_word, -1 AS sentiment_value;
```

²² The samples used are hosted at <https://bitbucket.org/syncfusiontech/hdinsight-succinctly/downloads/as-negative.csv> and positive.csv

²³ For full details, see the page [Opinion Mining, Sentiment Analysis, and Opinion Spam Detection](#) under the section Opinion Lexicon (or Sentiment Lexicon). The page is an excellent reference for some of the deeper aspects of Sentiment Analysis.

Finally, so we only have to operate against a single set of Sentiment words in downstream processing, we join the two relations together using a UNION statement:

```
sentiment = UNION positive, negative;
```

Next, we join our deconstructed messages and our Sentiment word lists. We will perform a Join similar to an Inner Join in T-SQL in that the output result set will only contain records where there has been a match. This will reduce the size of the output:

```
messages_joined = JOIN data_clean BY word, sentiment BY  
sentiment_word;
```

Here we have joined the relations `data_clean` and `positive_words` using the fields specified following the `BY` keyword. Because we have not modified the `JOIN` with any additional keywords (such as `LEFT` or `OUTER`), it performs an `INNER` join, discarding all rows where there is no match.

Again, at this point, no data has yet been processed.

Aggregating the Data

The next step is to aggregate the data and count the positive Sentiment. In Pig, grouping is a separate operation to performing aggregate functions such as `MIN`, `AVG` or `COUNT`, so first we must `GROUP` the data:

```
messages_grouped = GROUP messages_joined BY (year, month,  
day, message_id, author_id);
```

This produces a set of tuples in the `messages_grouped` relation for each year, month, day, and message id. Using the `DESCRIBE` keyword, we can see what this looks like in the Pig data structures:

```
DESCRIBE messages_grouped;
```

This produces the following description of the tuple in the `messages_grouped` relation:

```
messages_grouped: {group: (data_clean::year: chararray,  
data_clean::month: chararray, data_clean::day: chararray,  
data_clean::message_id: chararray, data_clean::author_id:  
chararray), messages_joined: {(data_clean::year: chararray,  
data_clean::month: chararray, data_clean::day: chararray,  
data_clean::message_id: chararray, data_clean::author_id:  
chararray, data_clean::word: chararray,  
sentiment::sentiment_word:chararray, sentiment::sentiment_value:  
int)}}}
```

However, this is a bit hard to read as is so, for illustrative purposes, we will restate it below, shortening the source relations names (from data_clean to dc, messages_joined to mj, and sentiment to s, respectively) and stripping out the data types:

```
messages_grouped: {  
  group: (dc::year, dc::month, dc::day, dc::message_id, dc::author_id),  
  mj: {(dc::year, dc::month, dc::day, dc::message_id, dc::author_id,  
    dc::word, s::sentiment_word, s::sentiment_value)}  
}
```

This creates two fields, one called “group” (highlighted green) which is a tuple that holds all the fields by which the relation is GROUPED. The second field (highlighted blue) is a bag that takes the name of the original relation (in this case, message_joined).

The second field will contain all the records that are associated with the unique set of keys within the first “group” field. For simple examples of this, please see the documentation.²⁴

Now that we have our GROUPED records, we need to count them:

```
message_sum_sentiment = FOREACH messages_grouped GENERATE  
group AS message_details,  
SUM(messages_joined.sentiment_value) AS sentiment;
```

This uses a FOREACH construct to generate new records using the “group” that was created by the GROUP operation, and performing an operation on the bag within that group. In this case, to perform a SUM operation.

Finally, to get a set of data we can export to a relational engine for further processing, we need to transform the record into a flat data structure that a relational structure can recognize:

```
message_sentiment_flat = FOREACH message_sum_sentiment  
GENERATE FLATTEN(message_details), (int)sentiment;
```

Here we use the FLATTEN command to un-nest the tuples created in the GROUP and COUNT operations.

Exporting the Results

Finally, we get to the stage where processing occurs:

²⁴ Pig GROUP documentation: <http://pig.apache.org/docs/r0.10.0/basic.html#GROUP>


```
STORE message_sentiment_flat INTO
'wasb://<container>@<storageaccount>.blob.core.windows.net/u
ser/hadoop/pig_out/messages' USING PigStorage('|');
```

The STORE command sends the content of a relation to the file system. Here we place the content of the FLATTENED relation “message_sentiment_flat” into Azure Blob Storage using the PigStorage function, specifying a pipe as the delimiter.

It would be possible to compress the output at this point should you choose to do so.²⁵

This causes all the relations in the chain to be processed and to populate the relation “message_sentiment_flat” for output so, in the command shell, MapReduce jobs can be seen to be initiated as follows:

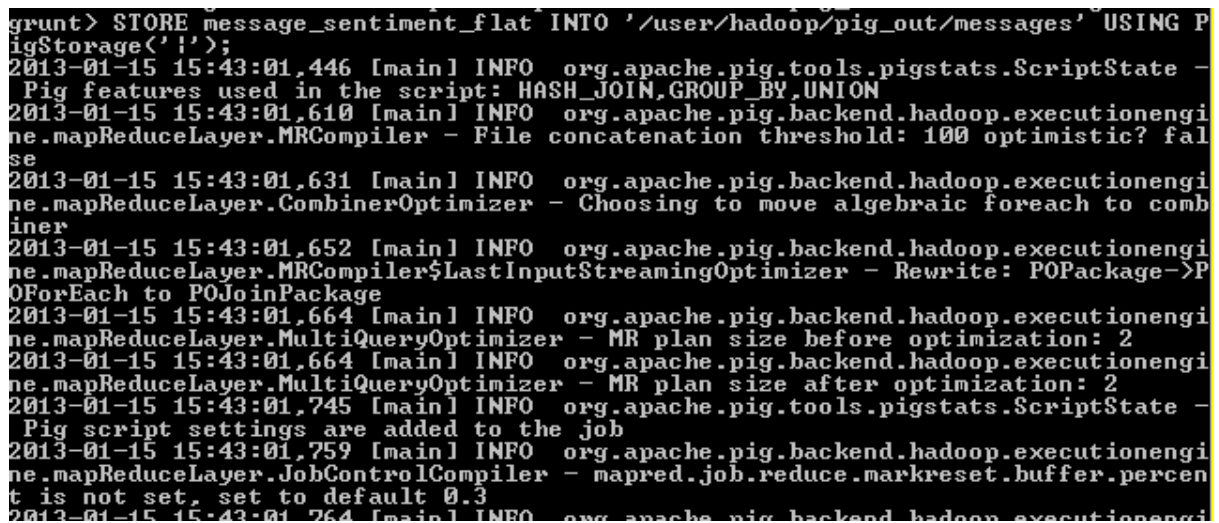


Figure 7: Pig command launching MapReduce jobs

This step completes the output of our analysis to Azure Blob Storage.

Additional Analysis on Word Counts

In addition to analysis at the message level, some aggregate analysis of Sentiment-loaded words was carried out. This will be looked at in less detail but will be referenced in the subsequent section on Hive.

First, we GROUP the data_clean relation by word so that we can then COUNT the word frequency:

```
words_group = GROUP data_clean BY (word);

words_count = FOREACH words_group GENERATE group AS words,
COUNT (data_clean) AS count;
```

²⁵ Compressing output in a Pig STORE command: <http://stackoverflow.com/questions/4968843/how-do-i-store-gzipped-files-using-pigstorage-in-apache-pig>

Next, we need to flatten the data and add Sentiment. Note that in the Sentiment join a LEFT join is used, so the complete list of words is retained (less those eliminated in the Mapper process):

```
words count flat = FOREACH words count GENERATE  
FLATTEN(words), (int)count;  
  
words count sentiment = JOIN words count flat BY words LEFT,  
sentiment BY sentiment_word;
```

Then we need to GROUP the records by the word, then aggregate the Sentiment using SUM, and count the word frequency using a COUNT function:

```
words count sentiment group = GROUP words count sentiment BY  
(words);  
  
words sum sentiment = FOREACH words count sentiment group  
GENERATE group AS words, SUM(words count sentiment.count) AS  
count, SUM(words count sentiment.sentiment value) AS  
sentiment;
```

Finally, we need to STORE the data to Azure Blob Storage for further analysis:

```
STORE words sum sentiment INTO  
'wasb://<container>@<storageaccount>.blob.core.windows.net/u  
ser/hadoop/pig_out/words' USING PigStorage('|');
```

Chapter 9 Using Hive to Store the Output

As per the official [Apache Hive project page](#):



***Hive** is a data warehouse system for Hadoop that facilitates easy data summarization, ad hoc queries, and the analysis of large data sets stored in Hadoop compatible file systems. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time, this language also allows traditional Map/Reduce programmers to plug in their custom Mappers and Reducers when it is inconvenient or inefficient to express this logic in HiveQL.*

Hive is a language that allows you to put a SQL-like structure on top of data stored as raw files such as delimited text, then query that data using the HiveQL. It is architected in such a way as to allow the query jobs to be massively parallelized using the MapReduce paradigm, as Hive commands are transformed into MapReduce jobs in order to execute. As with Pig, this is not exposed to the Hive programmer directly.

There are some overlaps in functionality with Pig and there are cases where either Hive or Pig can perform the same function. Which tool you use is a decision based on the tasks to be performed, the comfort of the developer with the given language plus, of course, which approach will prove more efficient from a performance point of view. Hive is often a better entry point than Pig for those more familiar with traditional SQL.

Creating an External Table to Reference the Pig Output

With Pig having stored the output of the processing into a file on HDFS, we can now put a semantic layer on top of it using Hive. This is done by creating an entity in Hive called an External Table which describes a table-like structure for a file.

A code sample for referencing the output of the word counting process is below:

```
CREATE EXTERNAL TABLE words (  
  word STRING,  
  counts INT,  
  sentiment INT  
)
```

```
ROW FORMAT DELIMITED

FIELDS TERMINATED BY '124'

STORED AS TEXTFILE

LOCATION '/user/hadoop/pig_out/words';
```

This SQL-like structure creates an “External Table” (that is, a reference to a file outside of the Hive environment). It then defines columns using the primitives available as data types. It then specifies the row format, field delimiter (as an ASCII number), its storage type (in this case, a text file), and, of course, its location.

This text file can now be referenced in a Hive table as if it were a normal table in the Hive environment. A sample query and results are below:

```
SELECT * FROM words

ORDER BY counts DESC

LIMIT 10
```

This query selects the top 10 words by frequency count from the word count output.

Once data is present in a Hive table, it can be accessed via ODBC and imported into a tool of choice.

It is worth noting that Hive is not the only way to expose data. Output from Pig or even base Hadoop jobs are simply text files that can be processed by any tool that can interpret the structure in which the output is defined. However, Hive has the advantage of being able to interactively filter the content by using a WHERE clause or add fields or calculations using HiveQL.

Chapter 10 Using the Microsoft BI Suite to Visualize Results

Processing the data on Hadoop is a significant part of the story. However, for users to extract value from it, they need to be able to manipulate and visualize it. The Microsoft BI suite—through the data modeling tool PowerPivot and the data visualization tool PowerView—enables this via the familiar Excel interface.

The Hive ODBC Driver and PowerPivot

Part of the toolset available to Microsoft Big Data users is the Hive ODBC driver. This allows an ODBC connection to a Hive database to extract the results into Excel either directly or as a data source in a PowerPivot model. In this example, we will be using the output of the results from prior steps as a data source for a PowerPivot model.

Installing the Hive ODBC Driver

The Hive driver needs to be installed on the client machine. The latest version of the driver is available from the Microsoft Download Center, and the location of the latest driver can be found on the HDInsight documentation.²⁶

Simply download and install the appropriate driver for the client machine.

Setting up a DSN for Hive

A prerequisite for connection to Hive is to set up a 64-bit System Data Source Name (DSN) that Excel can reference. Searching for ODBC brings up two apps:

²⁶ Connect Excel to HDInsight with the Microsoft Hive ODBC driver:
<http://www.windowsazure.com/en-us/manage/services/hdinsight/connect-excel-with-hive-odbc/>

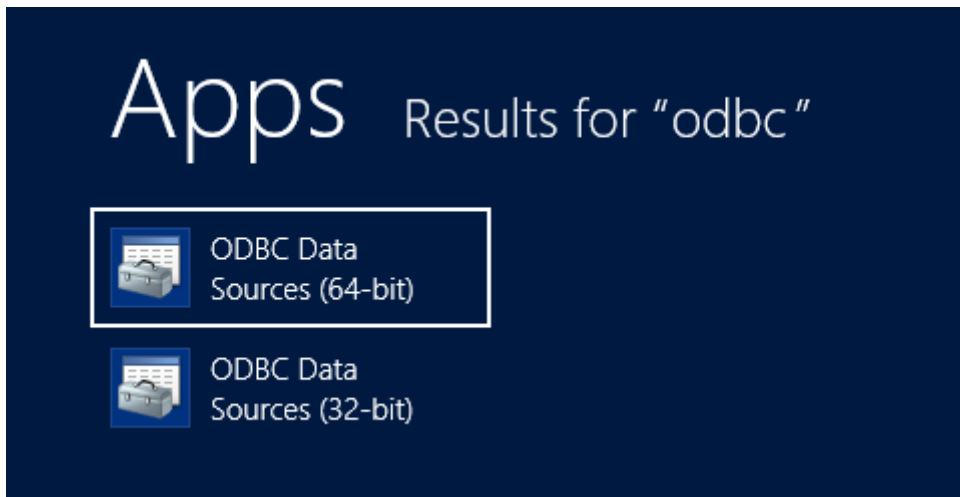


Figure 8: ODBC apps

From here, this launches the ODBC Data Source Administrator. Under System DSN, choose "Add". When choosing Create New Data Source, the Hive driver is listed as an available option:

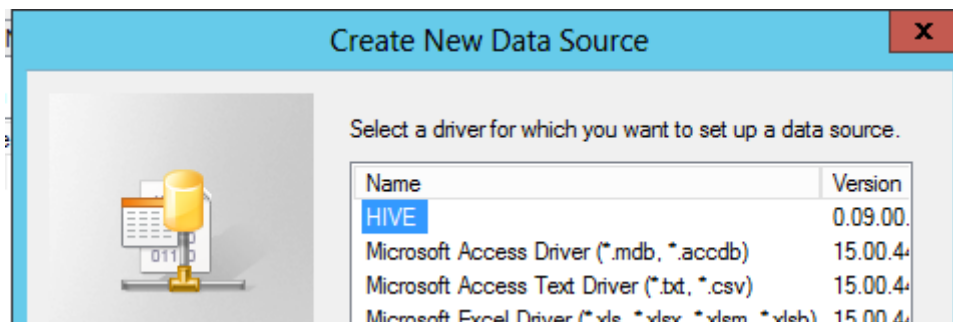


Figure 9: Creating a new System DSN using the Hive ODBC driver

Selecting it and choosing OK takes you to the Hive Data Source Configuration which needs to be updated to match the specific system settings:

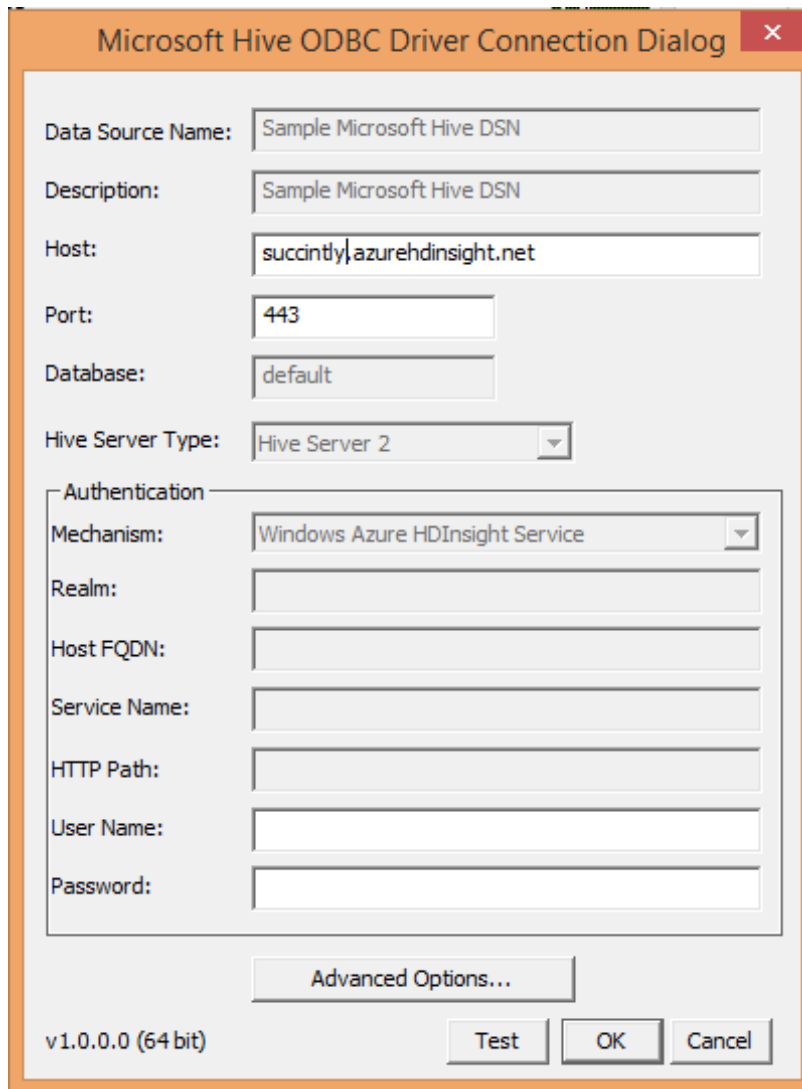


Figure 10: Configuring the Hive DSN

Once this step is complete, we can bring data into Excel.

Importing Data into Excel

Starting from within Excel, on the PowerPivot tab, select Manage Data Model:

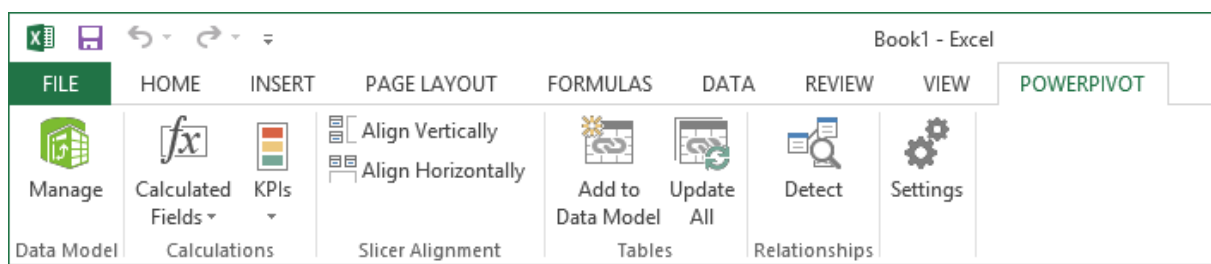


Figure 11: The Excel PowerPivot Ribbon tab

This will give you the option to Get External Data (in this case, “From Other Sources”):

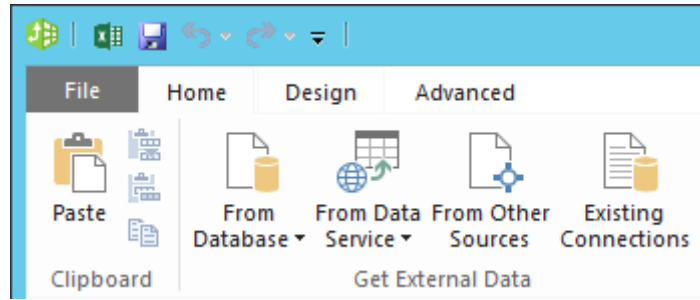


Figure 12: Excel PowerPivot Manage Data Model Ribbon

This opens up the Table Import Wizard. Under the “Relational Databases” section, select “Others (OLEDB/ODBC)” and click Next:

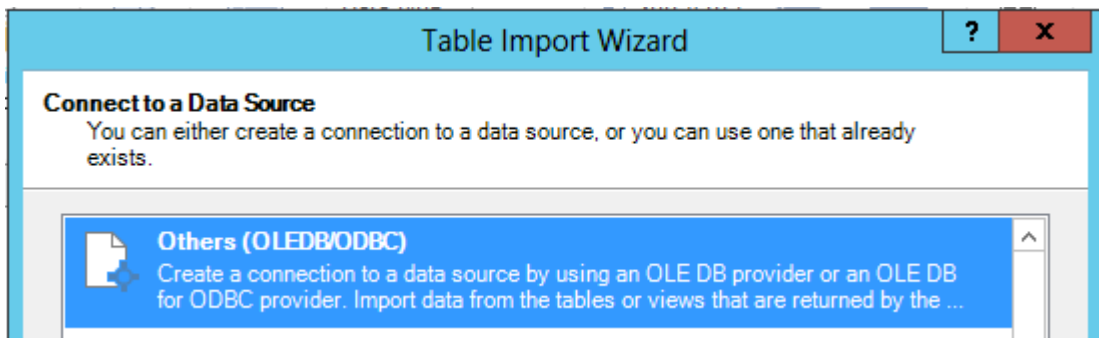


Figure 13: Excel PowerPivot Table Import Wizard - Data Source Type selection

This will then prompt you for a connection string. Choose the build option and, on the Provider tab, choose “Microsoft OLE DB Provider for ODBC Drivers”:

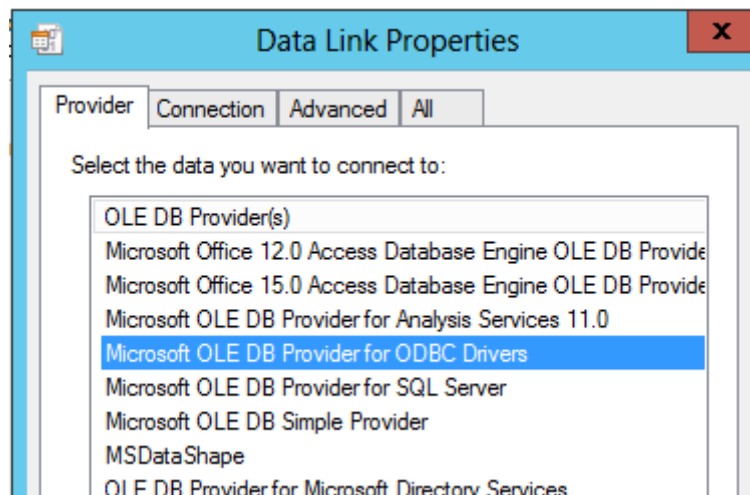


Figure 14: Excel PowerPivot Table Import Wizard - Data Link Type selection

Clicking Next will then get you on the Connection tab; you can then select the DSN created earlier. From there, click Next until the list of tables is displayed for import:

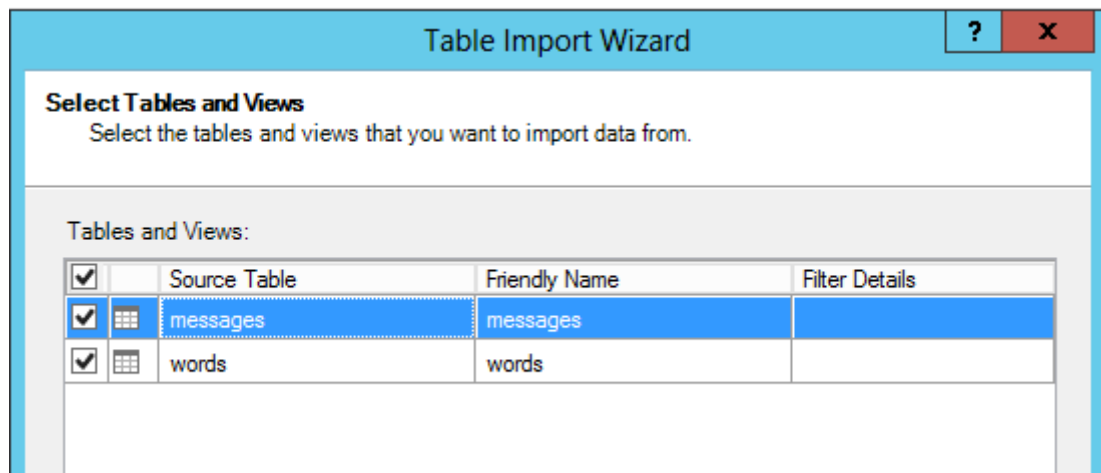


Figure 15: Excel PowerPivot Table Import Wizard - Selecting Hive tables

Select all relevant tables and choose Finish. The data from Hive is now available in PowerPivot for modeling and analysis as part of a normal Data Model:

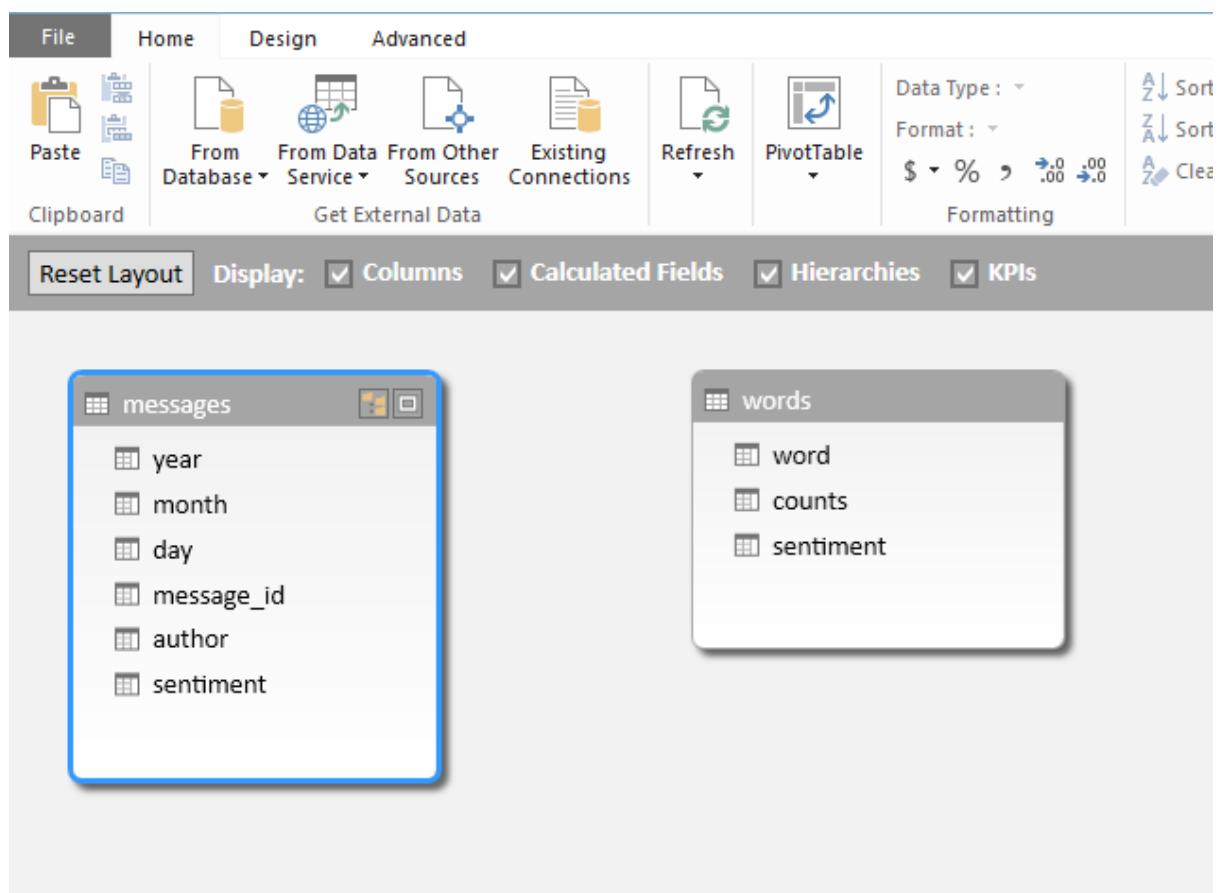


Figure 16: Excel PowerPivot Data Model Diagram View

Adding Context in PowerPivot

Part of PowerPivot's power is its ability to enrich your data with additional information from other sources (also known as "mashing up"), and by adding additional calculations and ways of analyzing measures. A few of these will be shown below.

Importing a Date Table from Windows Azure DataMarket

The data provides little scope for additional demographic or user data as it has been anonymized. However, it is possible to add some insight around time by adding a date table with a suitable hierarchy.

A BI-focused date table is available on Windows Azure DataMarket. To use this, when using the “Get External Data” option, choose “From Data Service” and “From Windows Azure Marketplace”:

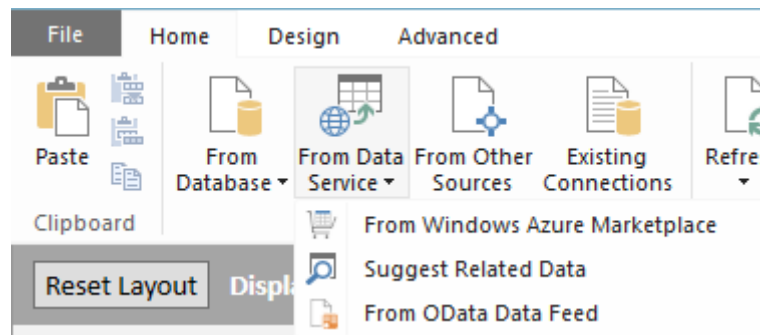


Figure 17: Excel PowerPivot Import Data from Data Service

This brings us to the Windows Azure Marketplace browser. A search for the “DateStream” set brings us to Boyan Penev’s date set, which is specifically designed for PowerPivot usage:

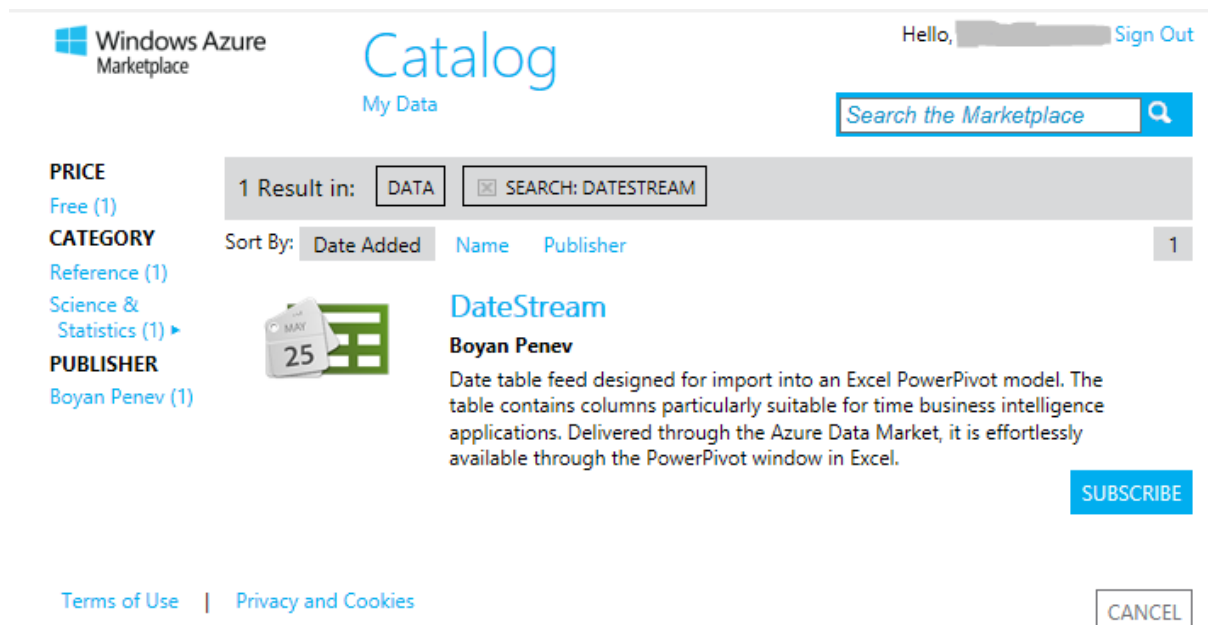


Figure 18: Excel Windows Azure Marketplace browser

Choose to Subscribe (a Windows Live account is required but the data set itself is free). Follow through the various steps until you reach the Select Query screen:

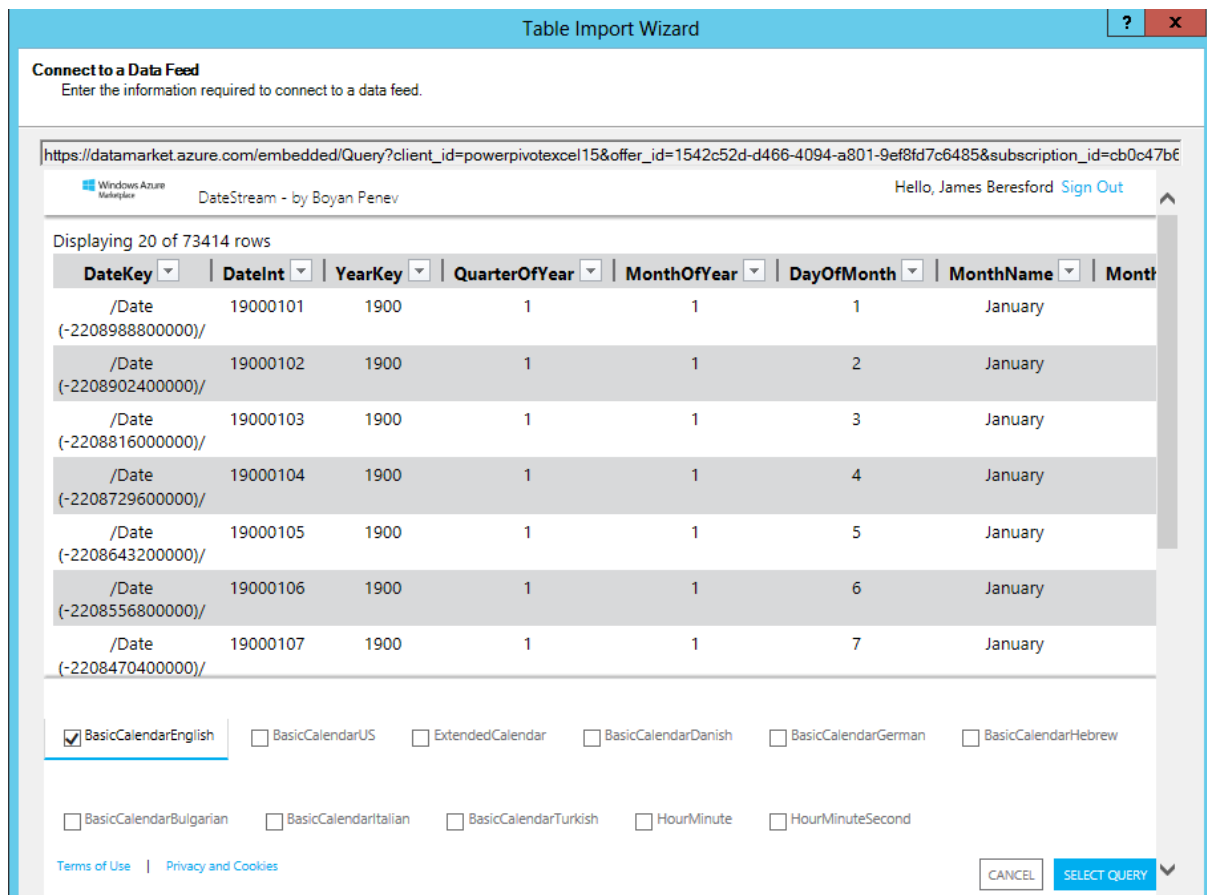


Figure 19: Excel Windows Azure Marketplace data feed options

Here, uncheck all options apart from “BasicCalendarEnglish” so that we limit the content retrieved to just what is relevant for our analysis.

Follow through the remaining steps in the process, leaving everything as default. The “BasicCalendarEnglish” table will now be added to the Data Model.

Creating a Date Hierarchy

To increase user friendliness, we will create a browse able calendar hierarchy on the Date table. From within the Data Model manager, right-click on the YearKey column in the “BasicCalendarEnglish” table and choose to “Create Hierarchy”:

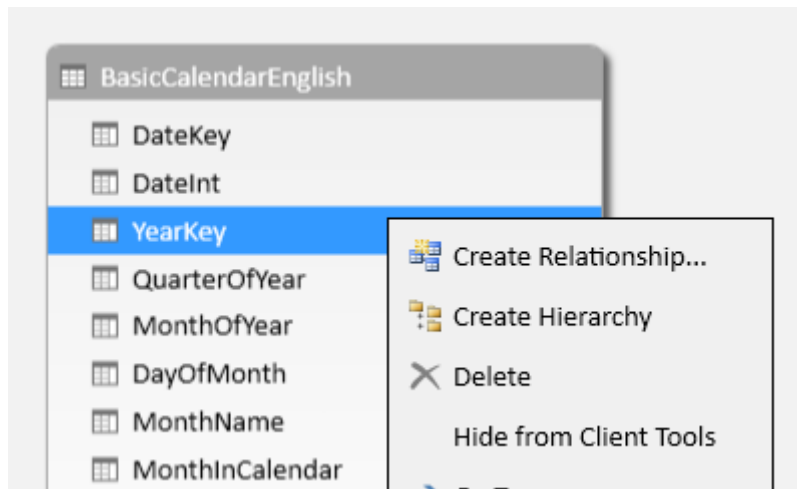


Figure 20: Excel PowerPivot Data Model - Creating a hierarchy

Give it the name Calendar. Then add the columns MonthName and DayOfMonth to the hierarchy by right-clicking and choosing “Add To Hierarchy” and selecting “Calendar”:

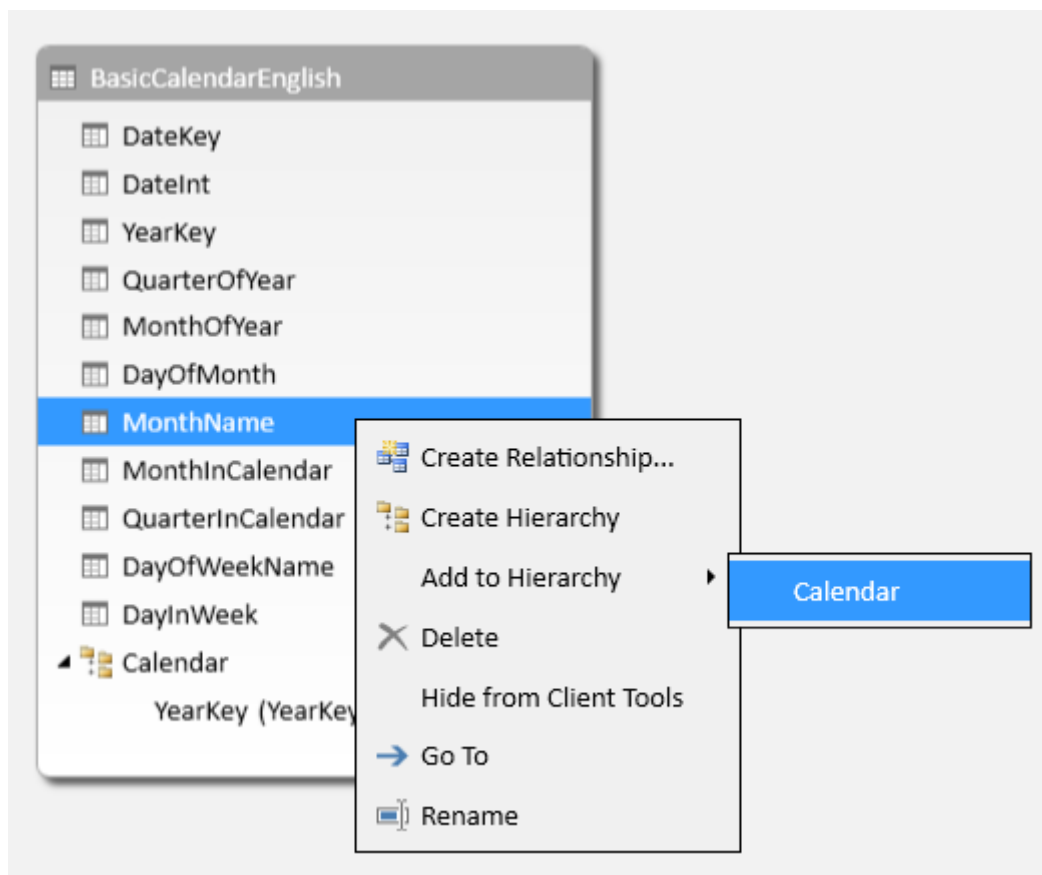


Figure 21: Excel PowerPivot Data Model - Adding levels to a hierarchy

A Calendar hierarchy has been created. PowerPivot is intelligent enough to determine the levels and tree without any user intervention.

Linking to the Sentiment Data

The final step is to connect the date table to the date-sensitive data in the model. To do this, we need to create a key on the data that will match the key on the date table. In this scenario, we will use the column “DateInt” which is an integer in the format YYYYMMDD.

From our data, we have three columns: Year, Month, and Day. First, we need to create a calculate column which will hold the data in a matching format. This can be done with a fairly simple Data Analysis Expressions (DAX)-calculated column that pads Month and Day with an extra zero as necessary, as shown below:

```
DateKey = ([year]*10000)+([month]*100)+[day]
```

Back in the Diagram View, dragging the DateInt column from the “BasicCalendarEnglish” table to the “messages” table creates the relationship between the two tables and allows viewing of the messages data using the Calendar hierarchy.

Adding Measures for Analysis

In order to examine aspects of the data such as count of posts, we need to add measures to the data that can be quickly assessed by the Data Model.²⁷

²⁷ MSDN PowerPivot measures documentation: <http://msdn.microsoft.com/en-us/library/gg399077.aspx>

year	month	day	message_id	author
2009	12	31	bd0e5490-ad93-4ff8-87eb-58dff9d3d835-307	a
2009	12	31	bd0e5490-ad93-4ff8-87eb-58dff9d3d835-391	a
2007	1	18	34cea9ba-e18a-405b-95da-be43371fd70f-13256	a
2005	10	6	4e4ca00f-078a-4f48-a7ee-0dc3018e9669-9412	a
2005	10	4	bd641ad4-c835-4873-95c2-b2225382f4b7-2064	a
2010	10	14	42a373be-7b29-4d79-97d7-fe05d9f5ae8a-11567	a
2005	10	6	752ce9ba-8d66-420f-aae7-e7e6555bce3d-5964	a
2009	12	31	a59dcae2-4897-4b2b-9b84-d5a7e7ec32a6-6503	a
2006	1	5	18c30c91-b6aa-4649-9914-cced213581b6-7607	a
2006	10	26	d92bf5ea-47f5-444f-9c25-0c7bf7458543-5104	a
2009	12	31	bd0e5490-ad93-4ff8-87eb-58dff9d3d835-423	a
2006	1	5	9174b10c-fad3-422a-8b51-9f0e97451947-10779	a
2009	12	31	bd0e5490-ad93-4ff8-87eb-58dff9d3d835-302	a
2009	12	31	bd0e5490-ad93-4ff8-87eb-58dff9d3d835-324	a
2009	12	31	a59dcae2-4897-4b2b-9b84-d5a7e7ec32a6-9096	a a a
2006	1	5	e5f87ac4-6cac-46d5-832b-c8298e1dd9da-8311	a a abthe wraith bb a a
2006	1	5	9174b10c-fad3-422a-8b51-9f0e97451947-2564	a a abthe wraith bb a a
2006	1	5	446abff4-942b-4d7c-99bf-36cb44d3aeb5-6877	a b
2006	1	5	446abff4-942b-4d7c-99bf-36cb44d3aeb5-6872	a b
2006	1	5	e5f87ac4-6cac-46d5-832b-c8298e1dd9da-7091	a b normal
2010	10	14	42a373be-7b29-4d79-97d7-fe05d9f5ae8a-8382	a baum
2010	10	14	0831ce57-8d69-45f9-ab9e-54ea6af8d8a3-9105	a baum
2010	10	14	0831ce57-8d69-45f9-ab9e-54ea6af8d8a3-9112	a baum

Figure 22: Adding a measure to the Data Model

This is done in the Data View. Clicking in a cell in the area under the data (arrow 1) enables us to enter a formula in the Formula bar (arrow 2).

From this we can enter a simple count measure using the DAX expression language:

```
Message Count:=COUNTA([message_id])
```

This uses the COUNTA function to count all non-empty cells—we know in this case all message_ids will be non-empty so this can be used to count all messages.²⁸

These measures are context-aware so, when using slicers, these will give the correct value depending on the slicer being used (for example, if you are looking at message count by users, it will give the count appropriate to that user).

More complex measures can be introduced such as this one which calculates average Sentiment per message:

```
Sentiment per Message:=sum([sentiment])/COUNTA([message_id])
```

This uses a simple sum function over the Sentiment, then divides by the message count to get the average Sentiment per message. This is context-aware so it could be used to determine the average Sentiment per user or sentiment trends over time.

Visualizing in PowerView

From within Excel, we can then visualize the data using the PowerView analytical tool against the Data Model we have created.

In Excel 2013, under the Insert tab on the Ribbon, we can launch “PowerView”:

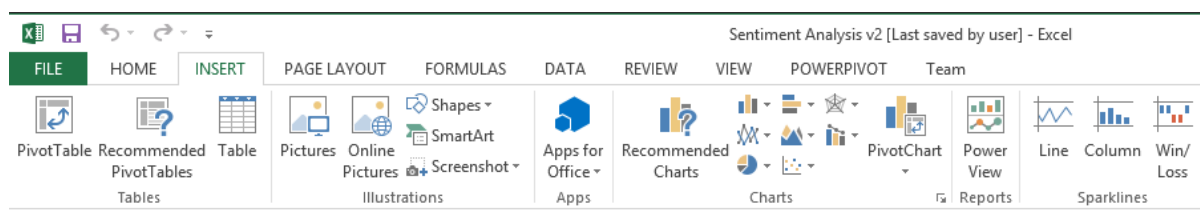


Figure 23: Launching PowerView in Excel

This then enables us to build reports from the Data Model we have created, selecting from the fields in the Model:

²⁸ DAX functions described: <http://social.technet.microsoft.com/wiki/contents/articles/684.powerpivot-dax-statistical-functions.aspx#counta>

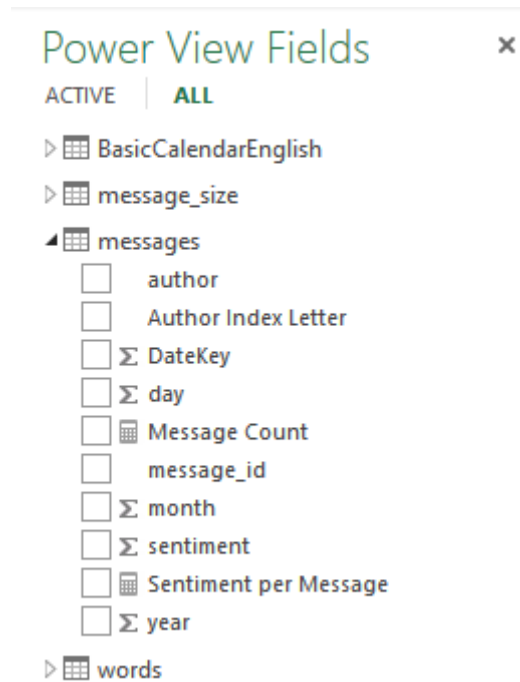


Figure 24: PowerView fields browsing

Through simple drag-and-drop operations, it becomes easy to create data visualizations with rich interactivity. Some examples are shown below.

The first example is a look at the distribution of Author Name (excluding “Unknowns”) by first letter of the author name. This highlights a high proportion of author names beginning with the letter “o”—an anomaly worth investigating further.

It features a bar chart showing the post counts by first letter of author name, and also a table listing the most prolific authors:

Author name distribution (exc. Unknown)

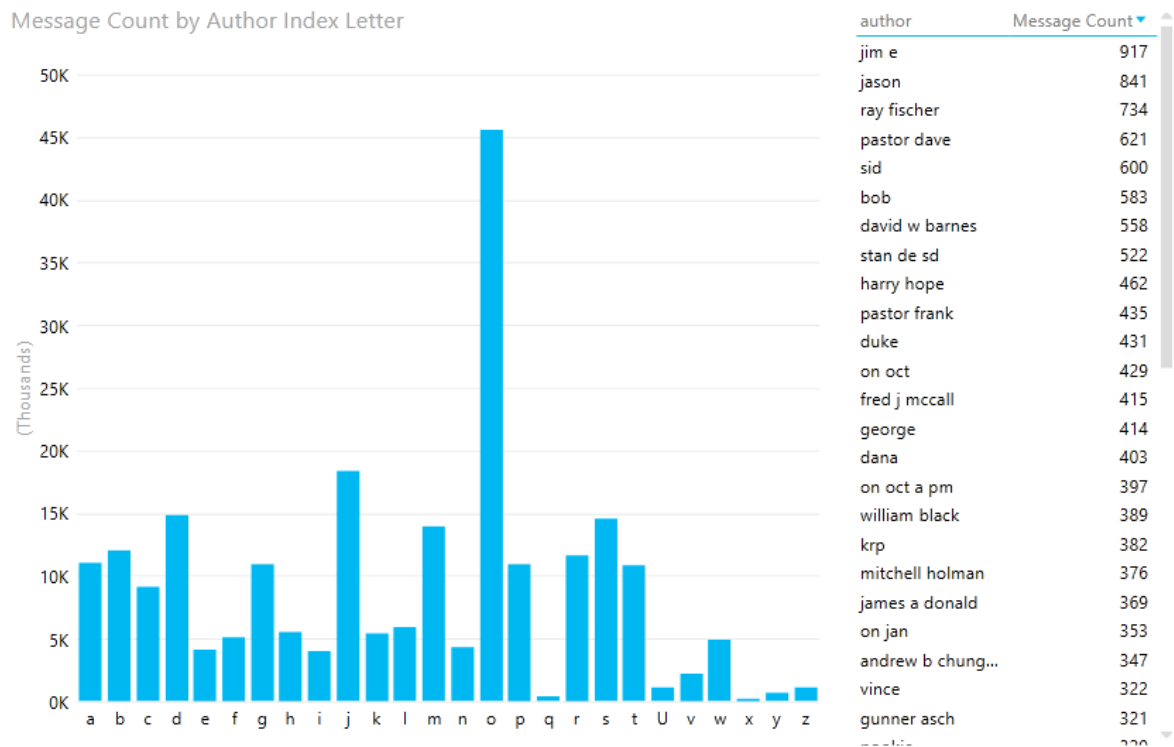


Figure 25: PowerView sample report "Author name distribution"

The next report looks at Sentiment related to Post Count, displayed as a scatter chart. Due to the number of data points, a warning appears indicating only a representative sample is being displayed.

The report appears to indicate that there is no strong relationship between post length and the overall Sentiment of the post:

Sentiment by Post Length

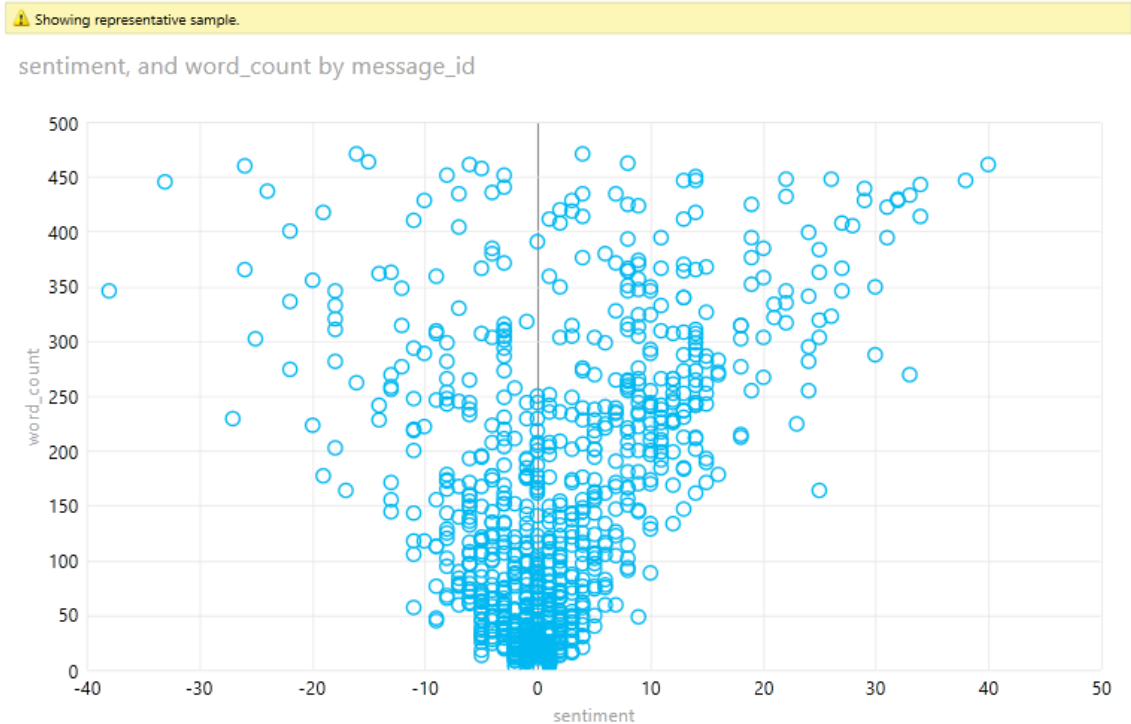


Figure 26: PowerView sample report "Sentiment by Post Length"

The final example shows "Sentiment by Author over Time". This can be filtered by author using the bar chart on the left, showing post count by author:

Sentiment by Author over Time

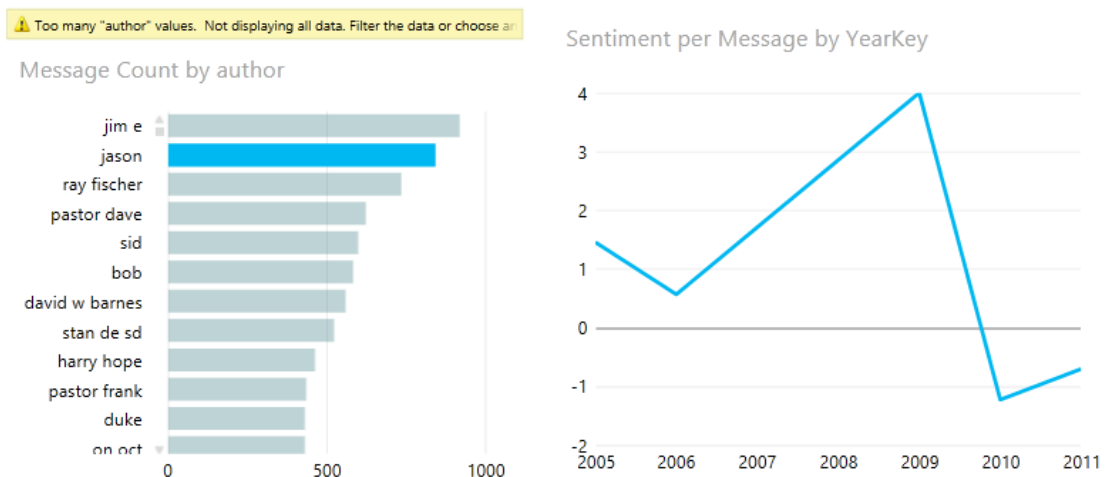


Figure 27: PowerView sample report "Sentiment by Author over Time"

PowerQuery and HDInsight

A second option for accessing data on HDInsight is to use the recently released Excel add-in PowerQuery.²⁹ This enables the direct extraction of text files from an HDInsight cluster or, strictly speaking, the Azure Blob Storage associated with the cluster.³⁰ In line with the architecture of HDInsight where compute is separate from storage, to access output from an HDInsight job, it is not necessary for the compute cluster to be up and running.

At present, this data access capability is not extended to PowerPivot, so the ability to bring in large volumes of data for analysis in Excel is limited.

²⁹ Download Microsoft Power Query for Excel: <http://office.microsoft.com/en-us/excel/download-data-explorer-for-excel-FX104018616.aspx>

³⁰ Connect Excel to Windows Azure HDInsight with Power Query: <http://www.windowsazure.com/en-us/manage/services/hdinsight/connect-excel-with-power-query/>

Other Components of HDInsight

There are three additional components of HDInsight that have not yet been addressed in detail, as they are more advanced features. These are likely to change and be extended as the platform evolves. Current details can be found on the documentation page that details what version of Hadoop is currently used in HDInsight.³¹

Oozie



Oozie is a graphical workflow engine that can run sequences of MapReduce and Pig jobs.

Oozie is exposed through REST APIs and the .NET SDKs.

For full details, see the official documentation.³²

Sqoop



Sqoop is a tool used to transfer data between Hadoop and relational databases using a JDBC driver. It is command line-based and allows the export of a single query or table's data from a relational source to either Hadoop as files or to Hive as tables. The reverse process allows the direct population of relational tables from Hadoop files.

For full details, see the official documentation.³³

Ambari



³¹ What's new in the cluster versions provided by HDInsight? <http://www.windowsazure.com/en-us/manage/services/hdinsight/versioning-in-hdinsight/>

³² Oozie Project page: <http://oozie.apache.org/index.html>

³³ Sqoop Project page: <http://sqoop.apache.org/>

Ambari is a framework for monitoring, managing, and provisioning clusters. It is still an incubator project as far as Apache is concerned.

Its exact role in HDInsight is unclear given the existing as well as planned capabilities in the Azure Management console, although it may have more relevance in the on-premises version when it arrives.

Like Oozie, it is exposed through REST APIs and the .NET SDKs.

For full details, see the official documentation.³⁴

³⁴ Ambari Project page: <http://incubator.apache.org/ambari/>