Getting started with Microsoft Azure HDInsight Service

HDInsight Service makes Apache Hadoop available as a service in the cloud. It makes the MapReduce software framework available in a simpler, more scalable, and cost efficient Microsoft Azure environment. HDInsight also provides a cost efficient approach to the managing and storing of data. HDInsight Service uses Microsoft Azure Blob Storage as the default file system.

In this tutorial, you will provision an HDInsight cluster using the Microsoft Azure Management Portal, deploy Java MapReduce programs using PowerShell, and then use Hive with HDInsight.

Prerequisites

Before you begin this hands on lab, you must have the following: - A Microsoft Azure subscription - sign up for a free trial - A computer that is running Windows 8, Windows 7, Windows Server 2012, or Windows Server 2008 R2.

Exercises

This hands-on lab includes the following exercises:

1. Set up local environment for running PowerShell
2. Provision an HDInsight cluster
3. Develop Java MapReduce programs for HDInsight
4. Use Hive with HDInsight

Estimated time to complete this lab: **30** minutes.

Excercise 1: Set up local environment for running PowerShell

In this exercise, you will use PowerShell Tools for Microsoft Azure HDInsight to run a MapReduce job. It requires the following configuration steps:

1. Install the Microsoft Azure module for Windows PowerShell
2. Install PowerShell Tools for Microsoft Azure HDInsight
3. Configure connectivity to your Microsoft Azure account

For more information, see Install and Configure PowerShell for HDInsight.

To install Microsoft Azure PowerShell

1. Open Internet Explorer, and browse to the Microsoft Azure Downloads page.
2. Under Windows downloads in the Command line tools section, click Microsoft Azure PowerShell, and then follow the instructions.

To install and import the PowerShell Tools for Microsoft Azure HDInsight

1. Open Internet Explorer, and then browse to Microsoft .NET SDK for Hadoop to download the package.
2. Click Run from the bottom of the page to run the installation package.
3. Open Microsoft Azure PowerShell. For instructions of opening a Microsoft Azure PowerShell console window, see Install and Configure PowerShell for HDInsight.

Your Microsoft Azure subscription information is used by the cmdlets to connect to your account. This information can be obtained from Microsoft Azure in a publishsettings file. The publishsettings file can then be imported as a persistent local config setting that the command-line interface will use for subsequent operations. You only need to import your publishsettings once.

> Notes: The publishsettings file contains sensitive information. It is recommended that you delete the file or take additional steps to encrypt the user folder that contains the file. On Windows, modify the folder properties or use BitLocker.

To download and import publishsettings

1. Sign in to the Microsoft Azure Management Portal using the credentials for your Microsoft Azure account.

2. Open Microsoft Azure PowerShell. For instructions of opening Microsoft Azure PowerShell console window, see Install and Configure

PowerShell for HDInsight.

3. Run the following command to download the publishsettings file.

   ```
   Get-AzurePublishSettingsFile
   ```

   The command opens an Internet Explorer window and a web page. The URL is *https://manage.Windowsazure.com/publishsettings/index?client=powershell*.

4. When prompted, download and save the publishing profile and note the path and name of the .publishsettings file. This information is required when you run the Import-AzurePublishSettingsFile cmdlet to import the settings. The default location and file name format is:

   ```
   C:\Users\<UserProfile>\Desktop\[MySubscription-…]-downloadDate-credentials.publishsettings
   ```

5. From the Microsoft Azure PowerShell window, run the following command to import the publishsettings file:

   ```
   Import-AzurePublishSettingsFile <PublishSettings-file>
   ```

   For example:

   ```
   Import-AzurePublishSettingFile "C:\Users\JohnDole\Desktop\Azure-8-30-2013-credentials.publishsettings"
   ```

6. Once the file is imported, you can use the following command to list your subscriptions:

   ```
   Get-AzureSubscription
   ```

7. When you have multiple subscriptions, you can use the following command to make a subscription the current subscription:

   ```
   Select-AzureSubscription -SubscriptionName <SubscriptionName>
   ```

Excercise 2: Provision an HDInsight cluster

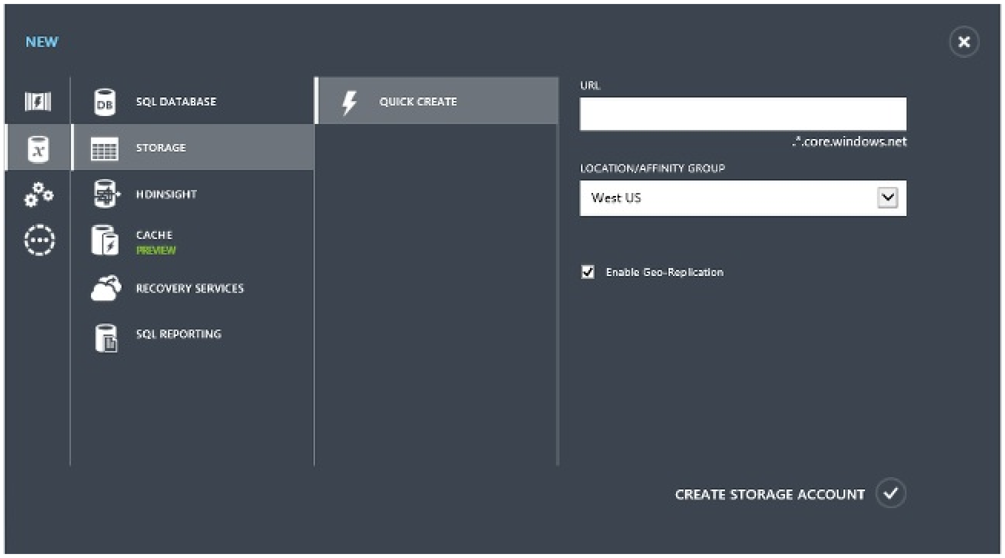
The HDInsight provision process requires a Microsoft Azure Storage account to be used as the default file system. The storage account must be located in the same data center as the HDInsight Service compute resources. Currently, you can only provision HDInsight clusters in the following data centers:

- US East
- US West
- Europe North

You must choose one of the two data centers for your Microsoft Azure Storage account.

To create a Microsoft Azure Storage account

1. Sign in to the Microsoft Azure Management Portal.
2. Click **NEW** on the lower left corner, point to **DATA SERVICES**, point to **STORAGE**, and then click **QUICK CREATE**.



URL

.*.core.windows.net

LOCATION/AFFINITY GROUP

West US

☑ Enable Geo-Replication

Create Storage Account

3. Enter **URL** and **LOCATION/AFFINITY** GROUP, and then click **CREATE STORAGE ACCOUNT**. You will see the new storage account in the storage list.

4. Wait until the **STATUS** of the new storage account is changed to **Online**.

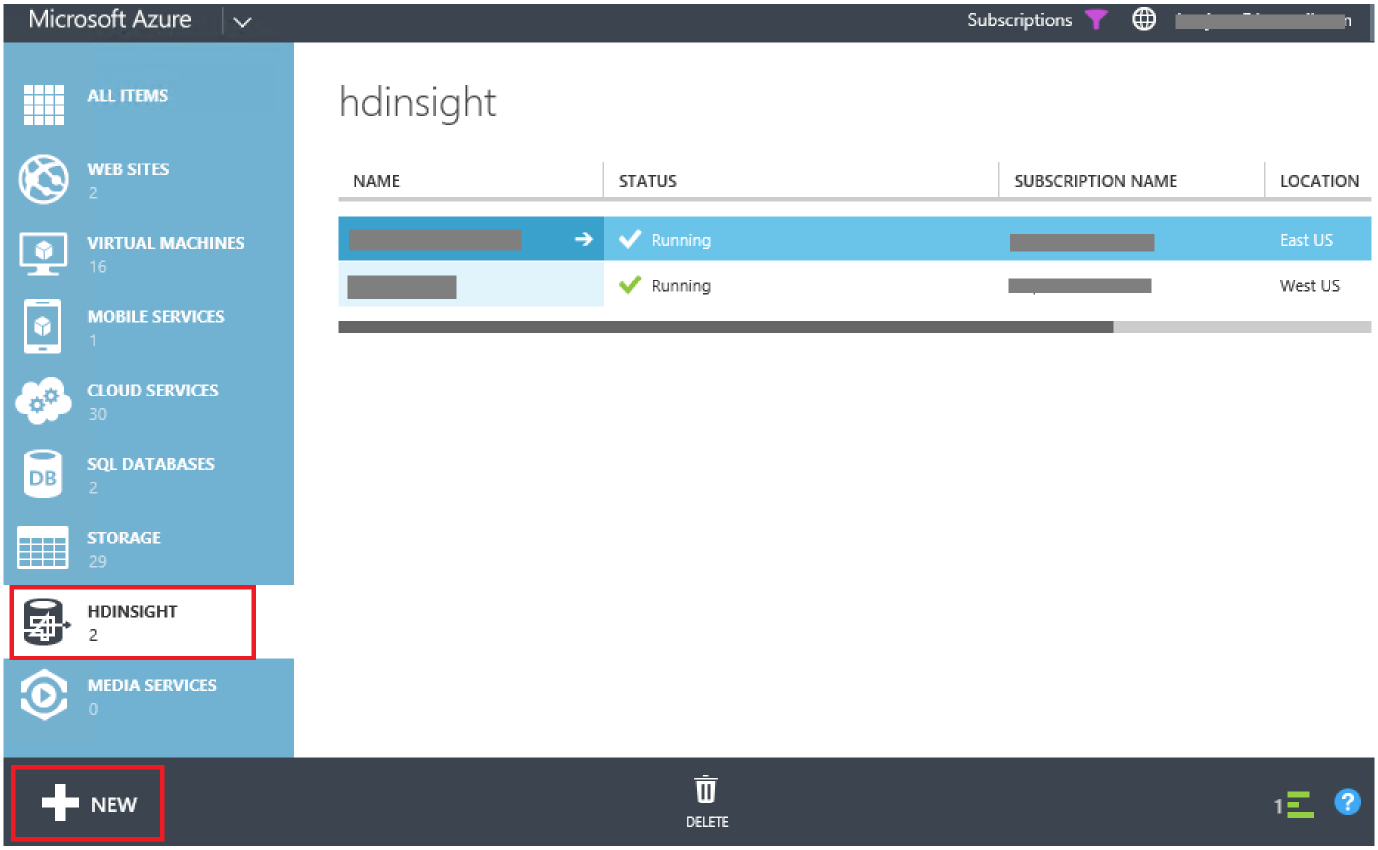
5. Click the new storage account from the list to select it.
6. Click **MANAGE ACCESS KEYS** from the bottom of the page.
7. Make a note of the **STORAGE ACCOUNT NAME** and the **PRIMARY ACCESS KEY**. You will need them later in the tutorial.

For the detailed instructions, see How to Create a Storage Account and Using Microsoft Azure Blob Storage with HDInsight.

To provision an HDInsight cluster

1. Sign in to the Microsoft Azure Management Portal.

2. Click **HDInsight** on the left to list the status of the clusters in your account. In the following screenshot, there is no existing HDInsight cluster.

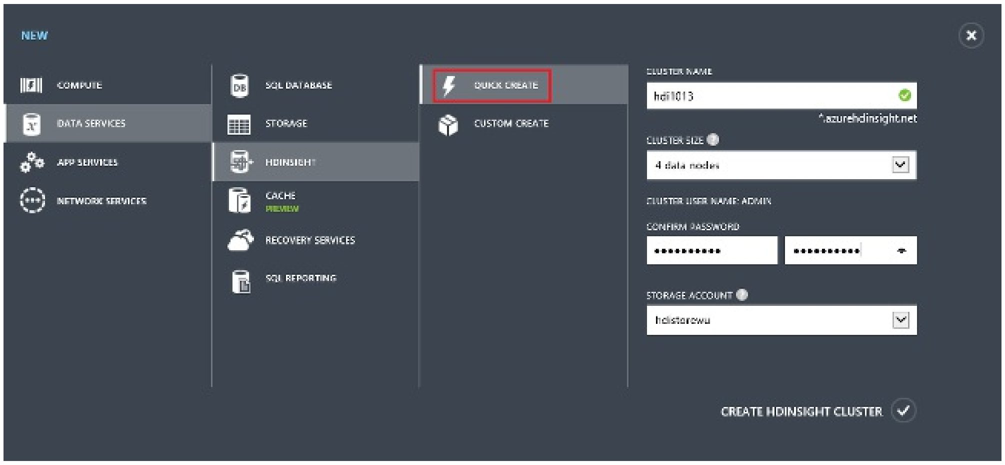


hdinsight

NAME	STATUS	SUBSCRIPTION NAME	LOCATION
[redacted]	✓ Running	[redacted]	East US
[redacted]	✓ Running	[redacted]	West US

Create HDInsight

3. Click **NEW** on the lower left side, click Data Services, click **HDInsight**, and then click **Quick Create**.



CLUSTER NAME: hdi1013 .azurehdinsight.net

CLUSTER SIZE: 4 data nodes

CLUSTER USER NAME: ADMIN

CONFIRM PASSWORD: •••••••••• ••••••••••

STORAGE ACCOUNT: hdistorewu

Quick Create HDInsight

4. Enter or select the following values:

   - **Cluster Name:** Name of the cluster
   - **Cluster Size:** Number of data nodes you want to deploy. The default value is 4. But 8, 16 and 32 data node clusters are also available on the dropdown menu. Any number of data nodes may be specified when using theCustom Createoption. Pricing details on the billing rates for various cluster sizes are available. Click the symbol just above the dropdown box and follow the link on the pop up.
   - **Password (cluster admin):** The password for the accountadmin. The cluster user name is specified to be "admin" by default when using the Quick Create option. This can only be changed by using the **Custom Create** wizard. The password field must be at least 10 characters and must contain an uppercase letter, a lowercase letter, a number, and a special character.
   - **Storage Account:** Select the storage account you created from the dropdown box. Once as Storage account is chosen, it cannot be changed. If the storage account is removed, the cluster will no longer be available for use. The HDInsight cluster location will be the same as the storage account.

5. Click **Create HDInsight Cluster** on the lower right. The cluster is now provisioned and when it will be available when its status is listed as

Running.

For information on using the **CUSTOM CREATE** option, see Provision HDInsight Clusters.

Excercise 3: Develop Java MapReduce programs for HDInsight

Develop a word counting MapReduce program in Java

1. Now you have an HDInsight cluster provisioned. The next step is to develop Java MapReduce programs and run it on your HDInsight. We will write a very simple program called **WordCount**. Word counting is a simple application that counts the occurrences of each word in a given input set. To write the word counting MapReduce job in Java, open Notepad or other editors and copy/paste the following program into the text editor.

```
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable>{
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

> Notice the package name is **org.apache.hadoop.examples** and the class name is **WordCount**. You will use the names when you submit the MapReduce job.

2. Save the file to any places on your computer with file name **WordCount.java**. Next you need to compile the program. If you are using windows, you can install HDInsight Emulator to compile and test your application. You can find more details from Get started with the HDInsight Emulator. If you are using Linux or Mac, you need to install Java SDK to compile your java code.

3. We presume that you have already installed Java SDK and we use the following command to comple the java code. The 2 jar files **commons-cli-1.2.jar** and **hadoop-core-1.1.0-SNAPSHOT.jar** can be found under **Azure-training-course\Day 2\6. Big Data analytics using Hadoop and SQL and no-SQL\hadoop\jar**.

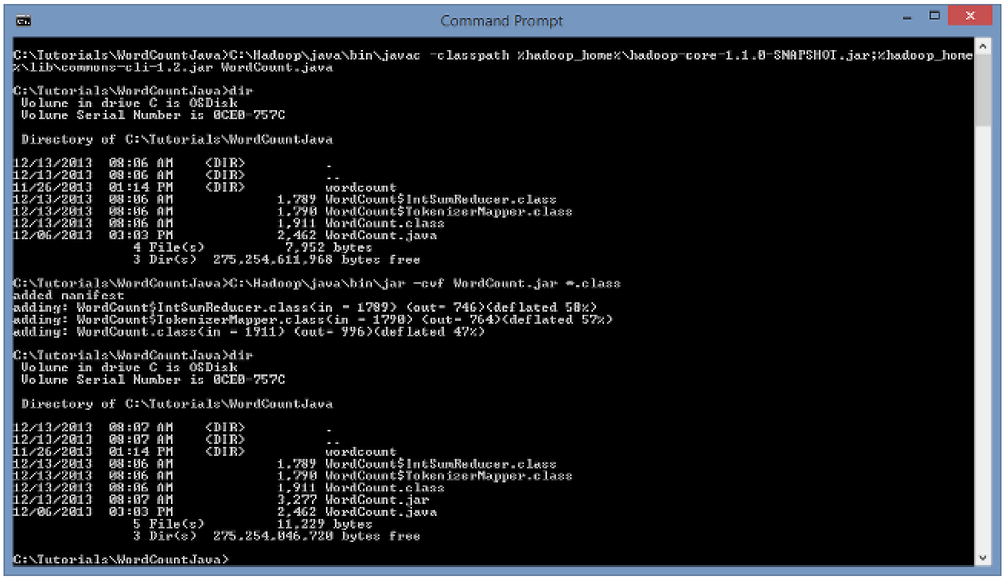
```
[JDK_FOLDER]\javac -classpath hadoop-core-1.1.0-SNAPSHOT.jar;commons-cli-1.2.jar WordCount.java
```

> Please ensure you add correct folder for [JDK_FOLDER] and **commons-cli-1.2.jar** and **hadoop-core-1.1.0-SNAPSHOT.jar** are in the same folder of your WordCount.java source code.

4. The compiler creates 3 class files in the current folder. Then we run the following command to create a jar file:

```
[JDK_FOLDER]\jar -cvf WordCount.jar *.class
```

The command creates a WordCount.jar file in the current folder.



```
C:\Tutorials\WordCountJava>C:\Hadoop\java\bin\javac -classpath %hadoop_home%\hadoop-core-1.1.0-SNAPSHOT.jar;%hadoop_home
%\lib\commons-cli-1.2.jar WordCount.java

C:\Tutorials\WordCountJava>dir
 Volume in drive C is OSDisk
 Volume Serial Number is 0CE0-757C

 Directory of C:\Tutorials\WordCountJava

12/13/2013  08:06 AM    <DIR>          .
12/13/2013  08:06 AM    <DIR>          ..
11/26/2013  01:14 PM    <DIR>          wordcount
12/13/2013  08:06 AM             1,789 WordCount$IntSumReducer.class
12/13/2013  08:06 AM             1,790 WordCount$TokenizerMapper.class
12/13/2013  08:06 AM             1,911 WordCount.class
12/06/2013  03:03 PM             2,462 WordCount.java
               4 File(s)          7,952 bytes
               3 Dir(s)  275,254,611,968 bytes free

C:\Tutorials\WordCountJava>C:\Hadoop\java\bin\jar -cvf WordCount.jar *.class
added manifest
adding: WordCount$IntSumReducer.class(in = 1789) (out= 746)(deflated 58%)
adding: WordCount$TokenizerMapper.class(in = 1790) (out= 764)(deflated 57%)
adding: WordCount.class(in = 1911) (out= 996)(deflated 47%)

C:\Tutorials\WordCountJava>dir
 Volume in drive C is OSDisk
 Volume Serial Number is 0CE0-757C

 Directory of C:\Tutorials\WordCountJava

12/13/2013  08:07 AM    <DIR>          .
12/13/2013  08:07 AM    <DIR>          ..
11/26/2013  01:14 PM    <DIR>          wordcount
12/13/2013  08:06 AM             1,789 WordCount$IntSumReducer.class
12/13/2013  08:06 AM             1,790 WordCount$TokenizerMapper.class
12/13/2013  08:06 AM             1,911 WordCount.class
12/13/2013  08:07 AM             3,277 WordCount.jar
12/06/2013  03:03 PM             2,462 WordCount.java
               5 File(s)         11,229 bytes
               3 Dir(s)  275,254,046,720 bytes free

C:\Tutorials\WordCountJava>
```

> If you are using Windows, you can install HDInsight Emulator to compile and test your application. You can find more details from Get started with the HDInsight Emulator.

Upload data to Azure Blob storage

Azure HDInsight uses Azure Blob storage for data storage. When an HDInsight cluster is provisioned, an Azure Blob storage container is used to store the system files. You can use either this default container or a different container (either on the same Azure storage account or a different storage account located on the same data center as the cluster) for storing the data files.

In this tutorial, you can use the storage account you created in Exercise 2 to save the data files and the MapReduce application. The data files are the text files in the **Azure-training-course\Day 2\6. Big Data analytics using Hadoop and SQL and no-SQL\hadoop\txt** directory on your workstation.

1. Open Azure PowerShell. Set the variables, and then run the commands:

```
$subscriptionName = "[AzureSubscriptionName]"
$storageAccountName_Data = "[AzureStorageAccountName]"
$containerName_Data = "[ContainerName]"
$location = "[MicrosoftDataCenter]"  # For example, "East US"
```

> The $subscripionName is associated with your Azure subscription. You must name the$storageAccountName*Data and $containerName*Data. For the naming restrictions, see Naming and Referencing Containers, Blobs, and Metadata.

2. Run the following commands to select the subscription and verify the storage account and the container:

```

# Select Azure subscription
Select-AzureSubscription $subscriptionName

Get-AzureStorageAccount -StorageAccountName $storageAccountName_Data
Get-AzureStorageContainer -Context $destContext
```

3. Then run the following commands to upload text files from the training txt directory.

```
$localFolder = "c:\Azure-training-course\Day 2\6. Big Data analytics using Hadoop and SQL and no-SQL\hadoop\txt"
$destFolder = "WordCount/Input"
$filesAll = Get-ChildItem $localFolder
# Copy the file from local workstation to the Blob container
foreach ($file in $filesAll){

    $fileName = "$localFolder\$file"
    $blobName = "$destFolder/$file"
    write-host "Copying $fileName to $blobName"
    Set-AzureStorageBlobContent -File $fileName -Container $containerName_Data -Blob $blobName -Context $destContext
}
```

> Notice the source file folder is the txt folder under *_Azure-training-course\Day 2\6. Big Data analytics using Hadoop and SQL and no-SQL\hadoop_*, and the destination folder is **WordCount/Input**.

4. Run the following command to list the uploaded files:

```
# List the uploaded files in the Blob storage container
Write-Host "The Uploaded data files:" -BackgroundColor Green
Get-AzureStorageBlob -Container $containerName_Data -Context $destContext -Prefix $destFolder
```

   You should see about 8 text data files.

5. Then we continue to upload the word counting application we compiled.

```
$jarFile = "Azure-training-course\Day 2\6. Big Data analytics using Hadoop and SQL and no-SQL\hadoop\src\WordCount.jar"
$blobFolder = "WordCount/jars"

Set-AzureStorageBlobContent -File $jarFile -Container $containerName_Data -Blob "$blobFolder/WordCount.jar" -Context $destCont

Write-Host "The Uploaded application file:" -BackgroundColor Green
Get-AzureStorageBlob -Container $containerName_Data -Context $destContext -Prefix $blobFolder
```

> Notice the destination folder is WordCount/jars and you should see the jar file listed there.

Run the MapReduce job on Azure HDInsight

Run the MapReduce job on Azure HDInsight

The following PowerShell script performs the following tasks:

1. provision an HDInsight cluster
    1. create a storage account that will be used as the default HDInsight cluster file system
    2. create a Blob storage container
    3. create an HDInsight cluster
2. submit the MapReduce job
    1. create a MapReduce job definition
    2. submit a MapReduce job
    3. wait for the job to complete
    4. display standard error
    5. display standard output
3. delete the cluster

    1. delete the HDInsight cluster
    2. delete the storage account used as the default HDInsight cluster file system
4. To run the PowerShell script, open Notepad and copy/paste the following code:

```
# The storage account and the HDInsight cluster variables
$subscriptionName = "[AzureSubscriptionName]"
$serviceNameToken = "[ServiceNameTokenString]"
$location = "[MicrosoftDataCenter]"    ###  must match the data storage account location
$clusterNodes = [NumberOFNodesInTheCluster]

$storageAccountName_Data = "[TheDataStorageAccountName]"
$containerName_Data = "[TheDataBlobStorageContainerName]"

$clusterName = $serviceNameToken + "hdicluster"

$storageAccountName_Default = $serviceNameToken + "hdistore"
$containerName_Default =  $serviceNameToken + "hdicluster"

# The MapReduce job variables
$jarFile = "wasb://$containerName_Data@$storageAccountName_Data.blob.core.windows.net/WordCount/jars/WordCount.jar"
$className = "org.apache.hadoop.examples.WordCount"
$mrInput = "wasb://$containerName_Data@$storageAccountName_Data.blob.core.windows.net/WordCount/Input/"
$mrOutput = "wasb://$containerName_Data@$storageAccountName_Data.blob.core.windows.net/WordCount/Output/"
$mrStatusOutput = "wasb://$containerName_Data@$storageAccountName_Data.blob.core.windows.net/WordCount/MRStatusOutput/"

# Create a PSCredential object. The username and password are hardcoded here.  You can change them if you want.
$password = ConvertTo-SecureString "Pass@word1" -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential ("Admin", $password)

Select-AzureSubscription $subscriptionName

#=============================
# Create a storage account used as the default file system
Write-Host "Create a storage account" -ForegroundColor Green
New-AzureStorageAccount -StorageAccountName $storageAccountName_Default -location $location

#=============================
# Create a Blob storage container used as teh default file system
Write-Host "Create a Blob storage container" -ForegroundColor Green
$storageAccountKey_Default = Get-AzureStorageKey $storageAccountName_Default | %{ $_.Primary }
$destContext = New-AzureStorageContext –StorageAccountName $storageAccountName_Default –StorageAccountKey $storageAccountKey_D

New-AzureStorageContainer -Name $containerName_Default -Context $destContext

#=============================
# Create an HDInsight cluster
Write-Host "Create an HDInsight cluster" -ForegroundColor Green
```

```
$storageAccountKey_Data = Get-AzureStorageKey $storageAccountName_Data | %{ $_.Primary }

$config = New-AzureHDInsightClusterConfig -ClusterSizeInNodes $clusterNodes |
    Set-AzureHDInsightDefaultStorage -StorageAccountName "$storageAccountName_Default.blob.core.windows.net" -StorageAccountKe
    Add-AzureHDInsightStorage -StorageAccountName "$storageAccountName_Data.blob.core.windows.net" -StorageAccountKey $storage

New-AzureHDInsightCluster -Name $clusterName -Location $location -Credential $creds -Config $config

#==============================
# Create a MapReduce job definition
Write-Host "Create a MapReduce job definition" -ForegroundColor Green
$mrJobDef = New-AzureHDInsightMapReduceJobDefinition -JobName mrWordCountJob  -JarFile $jarFile -ClassName $className -Argumen

#==============================
# Run the MapReduce job
Write-Host "Run the MapReduce job" -ForegroundColor Green
$mrJob = Start-AzureHDInsightJob -Cluster $clusterName -JobDefinition $mrJobDef
Wait-AzureHDInsightJob -Job $mrJob -WaitTimeoutInSeconds 3600

Get-AzureHDInsightJobOutput -Cluster $clusterName -JobId $mrJob.JobId -StandardError
Get-AzureHDInsightJobOutput -Cluster $clusterName -JobId $mrJob.JobId -StandardOutput

#==============================
# Delete the HDInsight cluster
Write-Host "Delete the HDInsight cluster" -ForegroundColor Green
Remove-AzureHDInsightCluster -Name $clusterName

# Delete the default file system storage account
Write-Host "Delete the storage account" -ForegroundColor Green
Remove-AzureStorageAccount -StorageAccountName $storageAccountName_Default
```

5. Set the first six variables in the script. $serviceNameToken will be used for the HDInsight cluster name, the default storage account name, and the default Blob storage container name. Because the service name must be between 3 and 24 characters, and the script append string with up to 10 character string to the names, you must limit the string to 14 or less characters. And the $serviceNameToken must use lower case. $storageAccountName*Data and $containerName*Data are the storage account and container that are used for storing the data files and the application. $location must match the data storage account location.

6. Review the rest of the variables.

7. Save the script file.
8. Open Azure PowerShell.
9. Run the following command to set the execution policy to remotesigned:

```
PowerShell -File <FileName> -ExecutionPolicy RemoteSigned
```

10. When prompted, enter username and password for the HDInsight cluster. Because you will delete the cluster at the end of the script and you will not need the username and password anymore, the username and password can be any strings. If you don't want to get prompted for the credentials, see Working with Passwords, Secure Strings and Credentials in Windows PowerShell.

Retrieve the MapReduce job output

This section shows you how to download and display the output. For the information on displaying the results on Excel, see Connect Excel to HDInsight with the Microsoft Hive ODBC Driver, and Connect Excel to HDInsight with Power Query.

1. Open Azure PowerShell window.
2. Change directory to **Azure-training-course\Day 2\6. Big Data analytics using Hadoop and SQL and no-SQL\hadoop**. The default Azure PowerShell folder is **C:\Windows\System32\WindowsPowerShell\v1.0**. The cmdlets you will run will download the output file to the current folder. You don't have permissions to download the files to the system folders.

3. Run the following commands to set the values:

```
$subscriptionName = "[AzureSubscriptionName]"
$storageAccountName_Data = "[TheDataStorageAccountName]"
```

```
$containerName_Data = "[TheDataBlobStorageContainerName]"
$blobName = "WordCount/Output/part-r-00000"
```

4. Run the following commands to create an Azure storage contect object:

```
Select-AzureSubscription $subscriptionName
$storageAccountKey = Get-AzureStorageKey $storageAccountName_Data | %{ $_.Primary }
$storageContext = New-AzureStorageContext –StorageAccountName $storageAccountName_Data –StorageAccountKey $storageAccountKey
```

5. Run the following commands to download and display the output:

```
Get-AzureStorageBlobContent -Container $containerName_Data -Blob $blobName -Context $storageContext -Force
cat "./$blobName" | findstr "there"
```

After the job is completed, you have the options to export the data to SQL Server or Azure SQL database using Sqoop, or to export the data to Excel.

Excercise 4: Use Hive with HDInsight

Apache Hive provides a means of running MapReduce job through an SQL-like scripting language, called *HiveQL*, which can be applied towards summarization, querying, and analysis of large volumes of data. In this article, you will use HiveQL to query the data in an Apache log4j log file and report basic statistics.

The Hive usage case

Databases are appropriate when managing smaller sets of data for which low latency queries are possible. However, when it comes to big data sets that contain terabytes of data, traditional SQL databases are often not an ideal solution. Database administrators have habitually scaling up to deal with these larger data sets, buying bigger hardware as database load increased and performance degraded.

Hive solves the problems associated with big data by allowing users to scale out when querying large data sets. Hive queries data in parallel across multiple nodes using MapReduce, distributing the database across on an increasing number of hosts as load increases.

Hive and HiveQL also offer an alternative to writing MapReduce jobs in Java when querying data. It provides a simple SQL-like wrapper that allows queries to be written in HiveQL that are then compiled to MapReduce for you by HDInsight and run on the cluster.

Hive also allows programmers who are familiar with the MapReduce framework to plug in custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the HiveQL language.

Hive is best suited for the batch processing of large amounts of immutable data (such as web logs). It is not appropriate for transaction applications that need very fast response times, such as database management systems. Hive is optimized for scalability (more machines can be added dynamically to the Hadoop cluster), extensibility (within the MapReduce framework and with other programming interfaces), and fault-tolerance. Latency is not a key design consideration.

Generally, applications save errors, exceptions and other coded issues in a log file, so administrators can use the data in the log files to review problems that may arise and to generate metrics that are relevant to errors or other issues like performance. These log files usually get quite large in size and contain a wealth of data that must be processed and mined for intelligence on the application.

Log files are therefore a paradigmatic example of big data. HDInsight provides a Hive data warehouse system that facilitates easy data summarization, ad-hoc queries, and the analysis of these big datasets stored in Hadoop compatible file systems such as Azure Blob Storage.

Upload data files to the Blob storage

HDInsight uses Azure Blob storage container as the default file system. For more information, see Use Azure Blob Storage with HDInsight.

In this demo, you use a log4j sample file distributed with the HDInsight cluster that is stored in *\example\data\sample.log*. Each log inside the file consists of a line of fields that contains a [LOG LEVEL] field to show the type and the severity. For example:

```
2012-02-03 20:26:41 SampleClass3 [ERROR] verbose detail for id 1527353937
```

To access files, use the following syntax:

```
wasb://[containerName]@[AzureStorageName].blob.core.windows.net
```

For example:

```
wasb://mycontainer@mystorage.blob.core.windows.net/example/data/sample.log
```

replace *mycontainer* with the container name, and *mystorage* with the Blob storage account name.

Because the file is stored in the default file system, you can also access the file using the following:

```
wasb:///example/data/sample.log
/example/data/sample.log
```

To generate your own log4j files, use the Apache Log4jlogging utility. For information on uploading data to Azure Blob Storage, see Upload Data to HDInsight.

Run Hive queries using PowerShell

In the last section, you uploaded a log4j file called sample.log to the default file system container. In this section, you will run HiveQL to create a hive table, load data to the hive table, and then query the data to find out how many error logs there were.

Hive queries can be run in PowerShell either using the **Start-AzureHDInsightJob** cmdlet or the **Invoke-Hive **cmdlet

To run the Hive queries using **Start-AzureHDInsightJob**, in PowerShell

1. Run the following command to connect to your Azure subscription:

   ```
   Add-AzureAccount
   ```

   You will be prompted to enter your Azure account credentials.

2. Set the variables in the following script and run it:

   ```
   # Provide Azure subscription name, and the Azure Storage account and container that is used for the default HDInsight file sys
   $subscriptionName = "[SubscriptionName]"
   $storageAccountName = "[AzureStorageAccountName]"
   $containerName = "[AzureStorageContainerName]"

   # Provide HDInsight cluster name Where you want to run the Hive job
   $clusterName = "[HDInsightClusterName]"
   ```

3. Run the following script to define the HiveQL queries:

   ```
   # HiveQL queries
   # Use the internal table option.
   $queryString = "DROP TABLE log4jLogs;" +
                  "CREATE TABLE log4jLogs(t1 string, t2 string, t3 string, t4 string, t5 string, t6 string, t7 string) ROW FORMAT
                  "LOAD DATA INPATH 'wasb://$containerName@$storageAccountName.blob.core.windows.net/example/data/sample.log' OVE
                  "SELECT t4 AS sev, COUNT(*) AS cnt FROM log4jLogs WHERE t4 = '[ERROR]' GROUP BY t4;"

   # Use the external table option.
   $queryString = "DROP TABLE log4jLogs;" +
                   "CREATE EXTERNAL TABLE log4jLogs(t1 string, t2 string, t3 string, t4 string, t5 string, t6 string, t7 string)
                   "SELECT t4 AS sev, COUNT(*) AS cnt FROM log4jLogs WHERE t4 = '[ERROR]' GROUP BY t4;"
   ```

   The LOAD DATA HiveQL command will result in moving the data file to the \hive\warehouse\ folder. The DROP TABLE command will delete the table and the data file. If you use the internal table option and want to run the script again, you must upload the sample.log file again. If you want to keep the data file, you must use the CREATE EXTERNAL TABLE command as shown in the script.

   You can also use the external table for the situation where the data file is located in a different container or storage account.

   Use the DROP TABLE first in case you run the script again and the log4jlogs table already exists.

4. Run the following script to create a Hive job definition:

```
# Create a Hive job definition
$hiveJobDefinition = New-AzureHDInsightHiveJobDefinition -Query $queryString
```

You can also use the -File switch to specify a HiveQL script file on HDFS.

5. Run the following script to submit the Hive job:

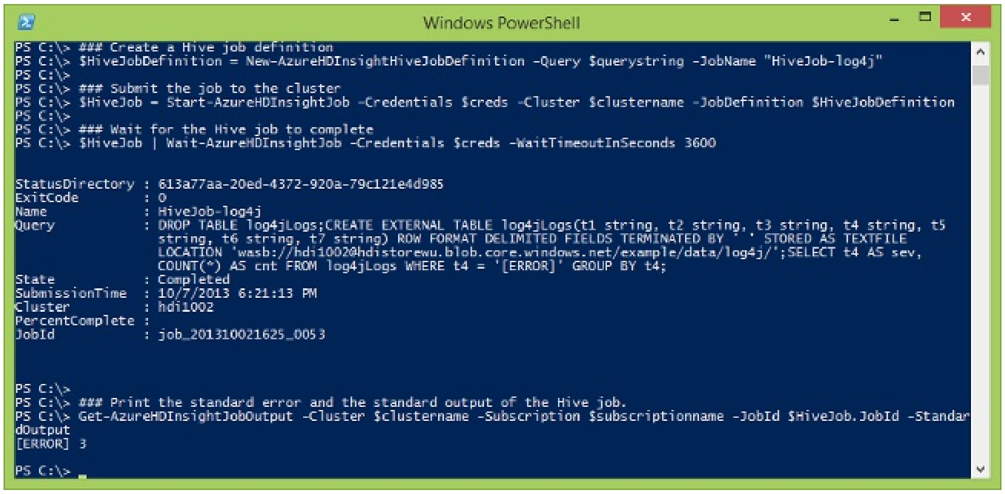
```
# Submit the job to the cluster
Select-AzureSubscription $subscriptionName
$hiveJob = Start-AzureHDInsightJob -Cluster $clusterName -JobDefinition $hiveJobDefinition
```

6. Run the following script to wait for the Hive job to complete:

```
# Wait for the Hive job to complete
Wait-AzureHDInsightJob -Job $hiveJob -WaitTimeoutInSeconds 3600
```

7. Run the following script to print the standard output:

```
# Print the standard error and the standard output of the Hive job.
Get-AzureHDInsightJobOutput -Cluster $clusterName -JobId $hiveJob.JobId -StandardOutput
```



```
PS C:\> ### Create a Hive job definition
PS C:\> $HiveJobDefinition = New-AzureHDInsightHiveJobDefinition -Query $querystring -JobName "HiveJob-log4j"
PS C:\>
PS C:\> ### Submit the job to the cluster
PS C:\> $HiveJob = Start-AzureHDInsightJob -Credentials $creds -Cluster $clustername -JobDefinition $HiveJobDefinition
PS C:\>
PS C:\> ### Wait for the Hive job to complete
PS C:\> $HiveJob | Wait-AzureHDInsightJob -Credentials $creds -WaitTimeoutInSeconds 3600


StatusDirectory : 613a77aa-20ed-4372-920a-79c121e4d985
ExitCode        : 0
Name            : HiveJob-log4j
Query           : DROP TABLE log4jLogs;CREATE EXTERNAL TABLE log4jLogs(t1 string, t2 string, t3 string, t4 string, t5
                  string, t6 string, t7 string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' STORED AS TEXTFILE
                  LOCATION 'wasb://hdi1002@hdistorewu.blob.core.windows.net/example/data/log4j/';SELECT t4 AS sev,
                  COUNT(*) AS cnt FROM log4jLogs WHERE t4 = '[ERROR]' GROUP BY t4;
State           : Completed
SubmissionTime  : 10/7/2013 6:21:13 PM
Cluster         : hdi1002
PercentComplete :
JobId           : job_201310021625_0053



PS C:\>
PS C:\> ### Print the standard error and the standard output of the Hive job.
PS C:\> Get-AzureHDInsightJobOutput -Cluster $clustername -Subscription $subscriptionname -JobId $HiveJob.JobId -Standar
dOutput
[ERROR] 3

PS C:\> _
```

Hive Powershell

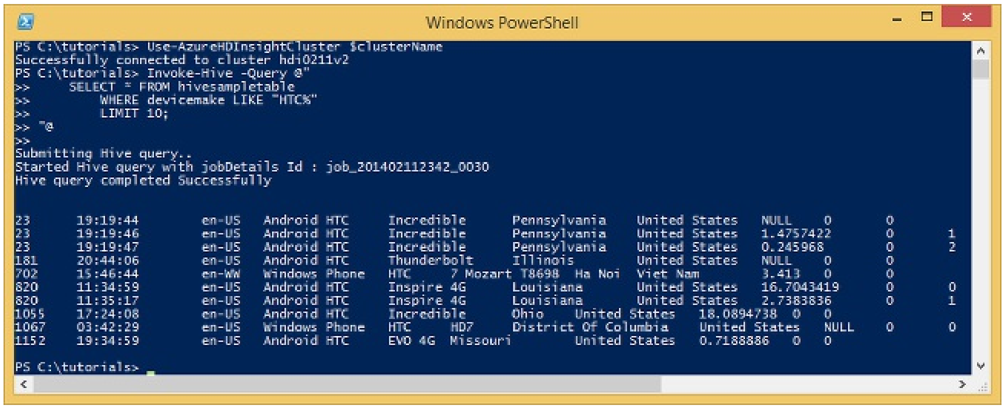
The results is:

```
[ERROR] 3
```

8. You can also submit Hive queries using **Invoke-Hive**, run the following script to invoke HiveQL queries:

```
Use-AzureHDInsightCluster $clusterName
$response = Invoke-Hive -Query @"
    SELECT * FROM hivesampletable
        WHERE devicemake LIKE "HTC%"
        LIMIT 10;
"@

Write-Host $response
```



```
PS C:\tutorials> Use-AzureHDInsightCluster $clusterName
Successfully connected to cluster hdi0211v2
PS C:\tutorials> Invoke-Hive -Query @"
>>      SELECT * FROM hivesampletable
>>          WHERE devicemake LIKE "HTC%"
>>          LIMIT 10;
>> "@
>>
Submitting Hive query..
Started Hive query with jobDetails Id : job_201402112342_0030
Hive query completed Successfully


23      19:19:44        en-US   Android HTC     Incredible      Pennsylvania    United States   NULL    0       0
23      19:19:46        en-US   Android HTC     Incredible      Pennsylvania    United States   1.4757422       0       1
23      19:19:47        en-US   Android HTC     Incredible      Pennsylvania    United States   0.245968        0       2
181     20:44:06        en-US   Android HTC     Thunderbolt     Illinois        United States   NULL    0       0
702     15:46:44        en-WW   Windows Phone   HTC     7 Mozart T8698  Ha Noi  Viet Nam        3.413   0       0
820     11:34:59        en-US   Android HTC     Inspire 4G      Louisiana       United States   16.7043419      0       0
820     11:35:17        en-US   Android HTC     Inspire 4G      Louisiana       United States   2.7383836       0       1
1055    17:24:08        en-US   Android HTC     Incredible      Ohio    United States   18.0894738  0   0
1067    03:42:29        en-US   Windows Phone   HTC     HD7     District Of Columbia    United States   NULL    0       0
1152    19:34:59        en-US   Android HTC     EVO 4G  Missouri        United States   0.7188886   0   0

PS C:\tutorials>
```

Hive Powershell Output

9. For longer HiveQL queries, it is recommended to use PowerShell Here-Strings or HiveQL script file. The following samples shows how to use the Invoke-Hive cmdlet to run a HiveQL script file. The HiveQL script file must be uploaded to WASB.

   ```
   Invoke-Hive -File "wasb://<ContainerName>@<StorageAccountName>/<Path>/query.hql"
   ```

   For more information about Here-Strings, see Using Windows PowerShell Here-Strings.

Summary

By completing this hands-on lab you learned the following:

1. Set up local environment for running PowerShell
2. Provision an HDInsight cluster
3. Develop Java MapReduce programs for HDInsight
4. Use Hive with HDInsight