Processing Big Data with Hadoop in Azure HDInsight

Lab 4C – Using the .NET Framework

Overview

In this lab, you will use the Microsoft .NET Framework to serialize and upload data to Azure storage, and to initiate Pig and Hive jobs.

What You'll Need

To complete the labs, you will need the following:

- A web browser
- A Microsoft account
- A Microsoft Azure subscription
- A Microsoft Windows computer with the following software installed:
  - Microsoft Azure PowerShell
  - Microsoft Visual Studio the Azure SDK
  - Microsoft Power BI Desktop
- The lab files for this course

Note: To set up the required environment for the lab, follow the instructions in the **Setup** document for this course. Specifically, you must have signed up for an Azure subscription, installed and configured Azure PowerShell, imported the publisher settings for your Azure subscription into PowerShell, installed Visual Studio and the Azure SDK, and installed Microsoft Power BI Desktop.

When working with cloud services, transient network errors can occasionally cause scripts to fail. If a script fails, and you believe that you have entered all of the required variables correctly; wait a few minutes and run the script again.

Provisioning HDInsight and Azure SQL Database

In this lab, you will build an application that collects sensor data from a racing car as it completes a test lap, and then processes the data in HDInsight before loading it to Azure SQL Database for further analysis. Before running the workflow, you need to provision the required Azure services.

Provision an Azure Storage Account and HDInsight Cluster

Note: If you already have an HDInsight cluster and associated storage account, you can skip this task.

1. In a web browser, navigate to http://azure.microsoft.com. Then click **Portal**, and if prompted, sign in using the Microsoft account that is associated with your Azure subscription.
2. In the Microsoft Azure portal, view the **HDInsight** page and verify that there are no existing HDInsight clusters in your subscription.
3. Click **NEW** (at the bottom of the page) and then click **CUSTOM CREATE**. Then use the New HDInsight Cluster wizard to create a new cluster with the following settings. Click the arrows to navigate through all of the wizard pages:
    - **Cluster Name**: *Enter a unique name (and make a note of it!)*
    - **Cluster Type**: Hadoop
    - **Operating System**: Windows Server 20012 R2 Datacenter
    - **HDInsight Version**: 3.2 (HDP 2.2, Hadoop 2.6)
    - **Data Nodes**: 2
    - **Region**: *Select any available region*
    - **Head Node Size**: A3 (4 cores, 7 GB memory)
    - **Data Node Size**: A3 (4 cores, 7 GB memory)
    - **HTTP User Name**: *Enter a user name of your choice (and make a note of it!)*
    - **HTTP Password**: *Enter and confirm a strong password (and make a note of it!)*
    - **Enable the remote desktop for cluster:** Selected
    - **RDP User Name:** *Enter another user name of your choice (and make a note of it!)*
    - **RDP Password:** *Enter and confirm a strong password (and make a note of it!)*
    - **Expires on**: Select tomorrow's date
    - **Enter the Hive/Oozie Metastore**: Unselected
    - **Storage Account**: Create New Storage
    - **Account Name**: *Enter a unique name for your storage account (and make a note of it!)*
    - **Default Container**: *Enter a unique name for your container (and make a note of it!)*
    - **Additional Storage Accounts**: 0
    - **Additional scripts to customize the cluster**: *None*
4. Wait for the cluster to be provisioned and the status to change to **Running** (this can take a while.)

Implementing Streaming MapReduce Code with .NET

Now that you have the required infrastructure, you are ready to implement a custom MapReduce solution by creating a streaming application based on the Microsoft .NET Framework. In this exercise, you will create a C# version of the WordCount solution you used in the first lab of this course.

Compile .NET Mapper and Reducer Assemblies

1. In the C:\HDILabs\Lab04C\WordCount folder, open **WordCount.sln** in Visual Studio. This solution contains two Microsoft C# projects for a console applications.
2. In Solution Explorer, In the **WordCountMapper** project, open **WordCountMapper.cs** and review the code it contains. This class implements the mapper for your MapReduce solution.
   Note that the code reads a stream of text data from standard input interface by using the **Console.Readline** method, splits the text based on a space delimiter, and writes the individual word strings to the standard output interface by using the **Console.WriteLine** method. These output represents each word in the source text as a key with a NULL value.
3. In Solution Explorer, in the **WordCountReducer** project, open **WordCountReducer.cs** and view the code it contains. This class implements the reducer for your MapReduce solution.
   Note that the code receives the keys generated by the mapper through the streaming interface by using the **Console.ReadLine** method, and then counts the instances of each word, writing the word and its count to the output interface by using the **Console.WriteLine** method.
4. On the **Build** menu, click **Build Solution** to compile the projects. Then close Visual Studio.

Upload the MapReduce Assemblies and Submit a Job

1. In the C:\HDILabs\Lab04C\WordCount folder, rename **Run MapReduce Job.txt** to **Run MapReduce Job.ps1** (you may need to modify the *View* options for the folder to see file extensions). Then right-click **MapReduce.ps1** and click **Edit** to open the script file in the PowerShell ISE.
2. Change the values assigned to the **$clusterName**, **$storageAccountName**, and **$containerName** variables to match the settings for your HDInsight cluster.
3. Examine the rest of the script, noting that it performs the following tasks:
    a) Removes any output left over from previous execution of the job.
    b) Uploads the contents of the **SourceData** subfolder to a folder named **/streamingexample/sourcedata** in your Azure blob container.
    c) Uploads the **WordCountmapper.exe** and **WordCountReducer.exe** assemblies you compiled in the previous procedure.
    d) Configures and submits a streaming MapReduce job that uses the assemblies you uploaded to process the source data files.
    e) Waits for the job to complete, and displays the job status output.
    f) Downloads the results file generated by the job to a local folder.
4. Save the script. Then on the toolbar, click **Run Script**.
5. Observe the information displayed in the PowerShell command line pane as the script runs. When the script finishes, the words and their counts are displayed.
6. Start Power BI Desktop and close the welcome page if it is displayed.

> **Note**: Power BI Desktop is the released version of the Power BI Designer preview tool used in the demonstrations for this course. The tool has been renamed and updated, and looks cosmetically different from the preview version; but still provides the same functionality as shown in the demonstrations.

7. In the **Get Data** drop-down list on the toolbar, click **More**.
8. In the **Get Data** dialog box, on the **File** page, click **Text**. Then click **Connect** and browse to the C:\HDILabs\Lab04C\WordCount\streamingexample\output folder. Change the file type to **All Files** so you can see the **part-00000** file, then open it. The data has been imported as delimited text.
9. Click **Edit** to open the query editor.
10. Right-click the column header for **Column 1**, click **Rename**, and rename the column to **Word**. Then rename **Column2** to **Count**.
11. With the **Count** column selected, on the toolbar, click the reverse sort (Z-A) button so that the most commonly used words are at the top of the list.
12. On the ribbon, click **Close & Load**. The data is loaded into the data model for the report.
13. Then in the **Fields** pane, expand the **part-00000** table. Then select **Word** and **Count**.
14. In the **Visualizations** pane, select **Bar Chart**. Then resize the chart to show the comparative frequency of the words in the source text.
15. Move the mouse to the top of the chart, click the **More Options** ellipse (…), and in the **Sort By** list, select **Count**.
16. After you have reviewed the results of the MapReduce job, close Power BI Desktop without saving any changes.

Implementing a .NET Client Application

In addition to implementing MapReduce code with the .NET Framework, you can create client applications that submit data and jobs to an HDInsight cluster. In this exercise, you will review a client application that

serializes and uploads simulated sensor readings taken from a racecar, and then submits Pig and Hive jobs to process the sensor data.

Add NuGet Packages to a .NET Application

1. In the C:\HDILabs\Lab04C\RaceTracker folder, open **RaceTracker.sln** in Visual Studio. This solution contains a Microsoft C# project for a console application.
2. On the **Tools** menu, point to **NuGet Package Manager**, and click **Manage NuGet Packages for Solution**.
3. In the NuGet Package Manager page, in the **nuget.org** package source, search for "Avro". Then install the latest stable version of the **Microsoft.Hadoop.Avro** package. Its dependencies are added automatically.
4. In the NuGet Package Manager page, in the **nuget.org** package source, search for "HDInsight". Then install the latest stable version of the **Microsoft.WindowsAzure.Management.HDInsight** package. Its dependencies are added automatically.
5. In the NuGet Package Manager page, change in the **Filter** list, select **Installed**, and select the **Microsoft.Hadoop.Client** package. Then in the **Action** drop-down list, select **Update**, select the **RaceTracker** project, and click **Update** to update this package to the latest stable build and install new dependencies. If prompted, accept the terms of the license agreement.
6. In the NuGet Package Manager page, search the list of installed packages for "Storage". If the **WindowsAzure.Storage** package is installed, in the **Action** drop-down list, select **Update**, select the **RaceTracker** project, and click **Update** to update this package to version **4.3.0** and install new dependencies. If prompted, accept the terms of the license agreement. If the **WindowsAzure.Storage** package is not installed, change the **Filter** to **All**, search for "Storage", and install version **4.3.0**.

Modify Configuration Settings

1. In Solution Explorer, open **App.config** to view the application configuration settings.
2. Change the value attributes of the following settings as necessary to match your HDInsight and Azure SQL Database configuration:
    - **StorageName**: The Azure storage account used by your HDInsight cluster.
    - **StorageKey**: Perform the following steps to obtain this:
        a) In Server Explorer, connect to your Azure subscription entering your Microsoft account credentials as if prompted.
        b) Expand **Azure** and **Storage**, and then right-click the storage account associated with your HDInsight cluster and click **Properties**.
        c) In the Properties pane, click the value **<Keys>** for the **Storage Account Keys** property, and then click the ellipsis (**...**) to display the keys.
        d) Click the icon next to the **Primary Key** to copy it to the clipboard. Then click **Close**.
        e) Paste the copied storage key into App.Config.
    - **ContainerName**: The Azure storage container where your HDInsight cluster stores files.
    - **ClusterName**: The name of your HDInsight cluster.
    - **UserName**: The HTTP user name for your HDInsight cluster.
    - **Password**: The password for your HTTP user.
3. Save App.config.

View Application Code

1. In Solution Explorer, open **Program.cs** and view the code in the **Main** function. The program performs the following tasks:
    - Reads (simulated) sensor data from a racing car every second until a lap is complete (in this deliberately simplistic simulation, the sensor data is read from a file).

- Each second, the **GetReadings** function gets the current GPS, engine, and brake sensor readings, adds them to a list for each reading type, and displays the reading values in the console window.
- After the lap has completed, the program creates a local file for each sensor (GPS, engine, and brake), and then calls the **SerializeReadings** function to serialize the reading lists to those files.
- After the data has been serialized to local files, the files are uploaded to Azure storage, together with Pig and Hive script files by the **UploadFiles** function.
- After the data and script files have been uploaded, the program calls the **SubmitJobs** function to submit Hadoop jobs to process the data.

2. Immediately below the **namespace RaceTracker** declaration (above the **Main** function), expand the **serialization classes** region to view the class declarations for the sensor reading objects used in the program. These classes include:
    - A **GpsReading** class that contains the following member properties:
        - **Time** (a string)
        - **Location** (a **Position** struct, which itself contains **lat** and **lon** properties)
        - **Speed** (a double precision decimal number)
    - An **EngineReading** class that contains the following member properties:
        - **Time** (a string)
        - **Revs** (a double precision decimal number)
        a) **OilTemp** (a double precision decimal number)
    - A **BrakeReading** class that contains the following member properties:
        - **Time** (a string)
        - **BrakeTemp** (a double precision decimal number)
3. View the **SerializeReadings** function. This function uses the **AvroContainer** class to compress and serialize the sensor reading object lists, including both the schema and the data.
4. View the **UploadFiles** function. This function uses a **CloudBlobClient** object to connect to your Azure storage container and upload the serialized data files. It also uploads the Pig and Hive script files in the **Scripts** subfolder.
5. In Solution Explorer, expand the **Scripts** folder and open the **ProcessData.pig** script file. Note that this Pig Latin script loads the data files using the **AvroStorage** deserializer, enabling it to work with the object schemas that were serialized in the Avro files. Then review the **CreateTables.hql** script, noting that this HiveQL code creates tables on the folders where the Pig Latin script stores its output, and creates a view to combine the sensor data.
6. In **Program.cs**, view the **SubmitJobs** function. This function uses a **PigJobCreateParameters** object to define a Pig job that runs the ProcessData.pig script, and then uses a **HiveJobCreateParameters** object to define a Hive job that runs the CreateTables.hql script. Both jobs are submitted using a **JobSubmissionClientFactory** object, and the **WaitForJobCompletion** function runs while each job is running to display a simple progress indicator in the console window.
7. Start the solution and observe the console window as the sensor readings are displayed. The program should run for around 90 seconds before serializing the captured readings, uploading the data and script files, and submitting the Pig and Hive jobs (which may take a while to run).

Install the Hive ODBC Provider

Note: If you already installed the Hive ODBC driver and created a data source name (DSN) when completing Lab 2A, you can skip this procedure.

1. Browse to http://www.microsoft.com/en-us/download/details.aspx?id=40886.
2. Download the installer that matches the 32-bit or 64-bit version of Windows installed on your computer (HiveODBC32.msi or HiveODBC64.msi).

3. Run the installer to install the Hive ODBC driver.
4. On the Start screen, view all apps. Then open the 32-bit or 64-bit ODBC Administrator tool (depending on the version of Windows you have installed) and add a System DSN based on the Microsoft Hive ODBC Driver with the following settings.
    - **Data Source Name**: Hive
    - **Description**: Hive tables in HDInsight
    - **Host**: The fully qualified DNS name of your HDInsight cluster (for example, *hd123456.azurehdinsight.net*)
    - **Port**: 443
    - **Database**: Default
    - **Hive Server Type**: Hive Server 2
    - **Authentication Mechanism**: Windows Azure HDInsight Service
    - **User Name**: The HTTP user name for your cluster
    - **Password**: The password for your HTTP user name

Visualize the Processed Data

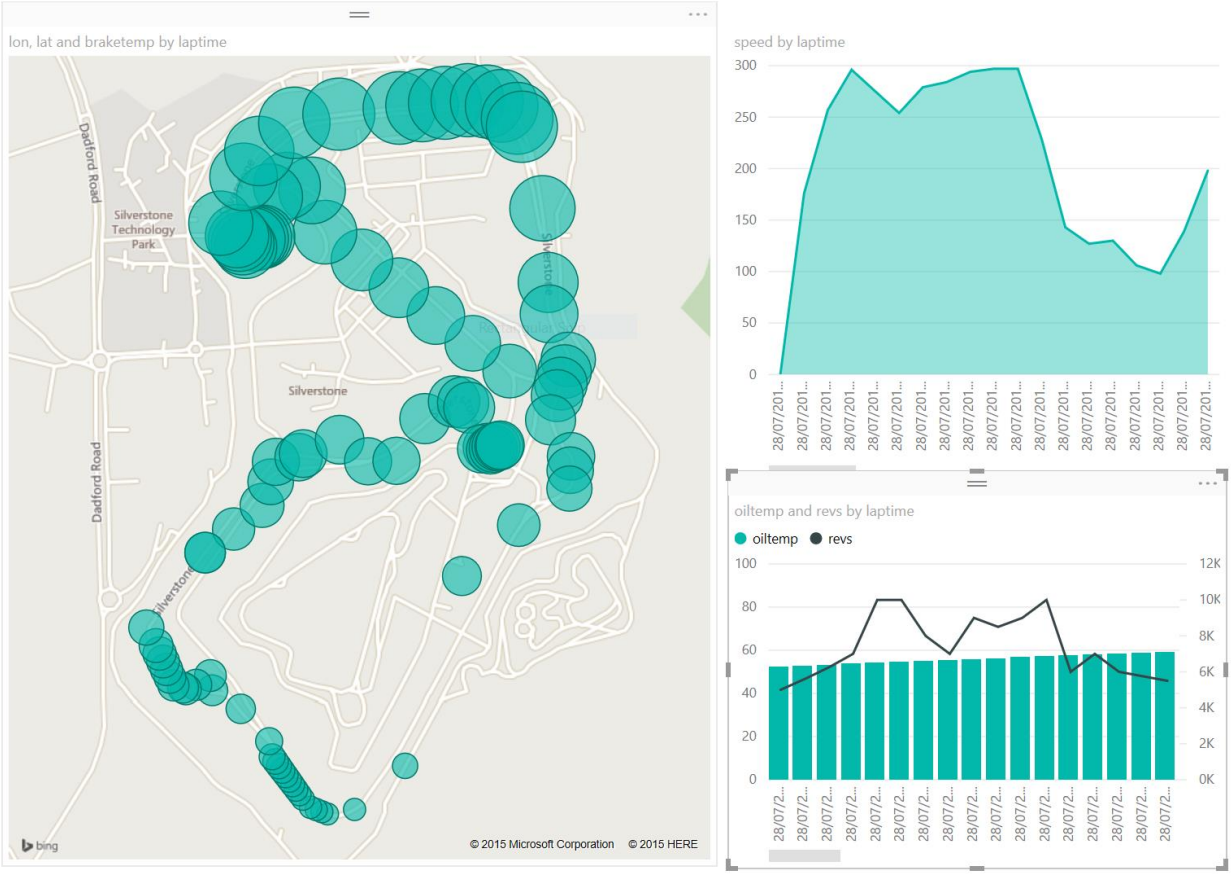
1. Start Power BI Desktop and close the welcome page if it opens.

    Note: Power BI Desktop is the released version of the Power BI Designer preview tool used in the demonstrations for this course. The tool has been renamed and updated, and looks cosmetically different from the preview version; but still provides the same functionality as shown in the demonstrations.

2. On the **Home** tab, in the **Get Data** list, click **More**.
3. In the **Get Data** page, click **Other**. Then select **ODBC** and click **Connect**.
4. In the **From ODBC** page, enter the following details, and then click **OK**.
    - Connection string: DSN=Hive;
    - SQL statement: SELECT * FROM lap ORDER BY laptime;
5. If you are prompted for credentials, on the **Database** page, enter the following details, and click **Connect**.
    - Username: The HTTP user name for your cluster
    - Password: The password for your HTTP user name
6. Wait for the data preview to be displayed (this may take a minute or so), and then click **Load**.
7. In the **Visualizations** pane, select **Map**, and resize the map to fill the left half of the report.
8. In the **Fields** pane, expand the **Query1** table, and drag the following fields to the specified areas in the Visualizations pane:
    - **laptime**: Location
    - **lat**: Latitude
    - **lon**: Longitude
    - **braketemp**: Values

    The map shows brake temperatures plotted on a map of a racing circuit, with larger circles indicating higher brake temperatures.

9. Select the empty area to the right of the map and, in the **Visualizations** pane, click **Area Chart**. Then, in the **Fields** pane, select the **laptime** and **speed** fields.
10. Select the empty area to the right of the map and under the area chart and, in the **Visualizations** pane, click **Line and Stacked Column Chart**. Then, in the **Fields** pane, select the **laptime** and **oiltemp** fields to display the oil temperature as columns, and drag the **revs** field to the **Line Values** area in the **Visualizations** pane to show the revs as a line. Then switch the visualization of this table to a **Stacked Column** chart and resize it to fill the bottom right quadrant of the report.
11. Your report should look like the following:



12. Click one of the brake temperature circles on the map and notice that the speed and oil temperature at that point in the lap are highlighted in the area and column charts (depending on which circle you selected, you may need to use the mini scroll bar under the column chart to see the highlighted rev value).
13. Choose the same circle again to remove the highlight.
14. Close Power BI Desktop without saving the report.

Cleaning Up

Now that you have finished this lab, you can delete the HDInsight cluster and storage account.

Delete the HDInsight Cluster

If you no longer need the HDInsight cluster and storage account used in this lab, you should delete them to avoid incurring unnecessary costs (or using credits in a free trial subscription).

1. In the Azure portal, click the **HDInsight** tab.
2. Select the row containing your HDInsight cluster, and then at the bottom of the page, click **Delete**. When prompted to confirm the deletion, click **Yes**.
3. Wait for your cluster to be deleted, and then click the **Storage** tab, and if necessary refresh the browser to view the storage account that was created with your cluster.
4. Select the row containing the storage account, and then at the bottom of the page, click **Delete**. When prompted to confirm the deletion, enter the storage account name and click **OK**.
5. Close the browser.