Microsoft

Processing Big Data with Azure Data Lake

Lab 3 – Using C# in U-SQL

Overview

U-SQL is designed to blend the declarative nature of SQL with the procedural extensibility of C#. In this lab, you will use C# code within U-SQL scripts.

What You'll Need

To complete this lab, you will need the following:

- A web browser
- A Microsoft account
- A Microsoft Azure subscription
- A Windows, Linux, or Mac OS X computer
- The lab files for this course
- An Azure Data Lake Analytics account

This lab includes an optional exercise that requires:

- A Windows computer
- Visual Studio 2015
- Visual Studio Azure Tools and SDK

Note: To set up the required environment for the lab, follow the instructions in the Setup document for this course.

Creating an Azure Data Lake Analytics Account

Note: If you completed the previous labs, and still have your Azure Data Lake Analytics account, you can skip this exercise and proceed straight to *Using Inline C# in a U-SQL Job*.

In this exercise, you will create an Azure Data Lake Analytics Account and associated Azure Data Lake store.

Note: The Microsoft Azure portal is continually improved in response to customer feedback. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing, but may not match the latest design of the portal exactly.

Create an Azure Data Lake Analytics Account

Before you can use Azure Data Lake Analytics to process data, you must create an Azure Data Lake Analytics account, and associate it with at least one Azure Data Lake store.

1. In a web browser, navigate to http://portal.azure.com, and if prompted, sign in using the Microsoft account that is associated with your Azure subscription.
2. In the Microsoft Azure portal, in the Hub Menu, click **New**. Then in the **Intelligence and analytics** menu, click **Data Lake Analytics**.
3. In the **New Data Lake Analytics Account** blade, enter the following settings, and then click **Create**:
    - **Name**: *Enter a unique name (and make a note of it!)*
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Create a new resource group with a unique name*
    - **Location:** *Select any available region*
    - **Data Lake Store:** *Create a new Data Lake Store with a unique name (and make a note of it!)*
    - **Pin to dashboard**: *Not selected*
4. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the resources to be deployed (this can take a few minutes.)

Upload Source Data Files

In this lab, you will use Azure Data Lake Analytics to process web server log data.

1. In the folder where you extracted the lab files, open the **iislogs** folder and then use a text editor to view the **2008-01.txt** file.
2. Review the contents of the file, noting that it contains some header rows (prefixed with a # character) and some space-delimited web server request records for the month of January in 2008. Then close the file without saving any changes. The other files in this folder contain similar data for February to June 2008.
3. In the Azure portal, view the **Data Explorer** page for your Azure Data Lake Analytics account, and create a new folder named **iislogs** in the root of your Azure Data Lake store.
4. Open the newly created **iislogs** folder. Then click **Upload**, select all of the files in the local **iislogs** folder (you can hold the CTRL key to select multiple files) and upload them.
5. Repeat the previous step to upload the **2008-07.txt** file from the local **July** folder to the **iislogs** folder in your Azure Data Lake store.

Create a Database

Creating a database enables you to store data in a structured format, ready to be queried by jobs.

1. In the Azure portal, on the blade for your Azure Data Lake Analytics account, click **New Job**.
2. In the **New U-SQL Job** blade, in the **Job Name** box, type **Create DB**.
3. In the code editor, enter the following code:

```
CREATE DATABASE IF NOT EXISTS webdata;

USE DATABASE webdata;

CREATE SCHEMA IF NOT EXISTS iis;

CREATE TABLE iis.log
(date string,
 time string,
 client_ip string,
 username string,
 server_ip string,
 port int,
 method string,
 stem string,
 query string,
 status string,
 server_bytes int,
 client_bytes int,
 time_taken int?,
 user_agent string,
 referrer string,
INDEX idx_logdate CLUSTERED (date)
PARTITIONED BY HASH (date));


@log =
EXTRACT date string,
        time string,
        client_ip string,
        username string,
        server_ip string,
        port int,
        method string,
        stem string,
        query string,
        status string,
        server_bytes int,
        client_bytes int,
        time_taken int?,
        user_agent string,
        referrer string
FROM "/iislogs/{*}.txt"
USING Extractors.Text(' ', silent:true);

INSERT INTO iis.log
SELECT * FROM @log;

CREATE VIEW iis.summary
AS
SELECT date,
       COUNT(*) AS hits,
       SUM(server_bytes) AS bytes_sent,
       SUM(client_bytes) AS bytes_received
FROM iis.log
GROUP BY date;
```

4. Click **Submit Job**, and observe the job status as it runs.
5. When the job has finished running, return to the blade for your Azure Data Lake Analytics account and click **Data Explorer**.
6. In the **Data Explorer** blade, under **Catalog**, verify that the **webdata** database is now listed (alongside the default **master** database).

Using Inline C# in a U-SQL Job

The simplest way to use C# in a U-SQL job is to include inline calls to C# functions in your U-SQL code.

Use a C# Function in a Query

IP addresses can be one of several types or *families*. Most IP addresses in use on the Internet belong to the *InterNetwork* (IPv4) family (for example *192.0.2.1*) or the *InterNetworkV6* (IPv6) family (for example *2001:db8:85a3:8d3:1319:8a2e:370:7348*). In this procedure, you will use C# code to determine the IP address family of client IP addresses by examining the **AddressFamily** property of the Microsoft .NET **System.Net.IPAddress** class.

1. In the Azure portal, on the blade for your Azure Data Lake Analytics account, click **New Job**.
2. In the **New U-SQL Job** blade, in the **Job Name** box, type **Get IP Details**.
3. In the code editor, enter the following code:

   ```
   USE DATABASE webdata;

   @ipdetails = SELECT DISTINCT client_ip,
           System.Net.IPAddress.Parse(client_ip).AddressFamily.ToString()
              AS ipfamily
   FROM iis.log;

   OUTPUT @ipdetails
       TO "/output/ipdetails.csv"
       USING Outputters.Csv();
   ```

   This code queries the **iis.log** table to return each distinct **client_ip** address and the IP family to which the address belongs.

4. Click **Submit Job** and observe the job details as it is run.
5. When the job has finished, click the **Output** tab and select **ipdetails.csv** to see a preview of the results (all the **client_ip** addresses should belong to the *InterNetwork* family).

Use a More Complex C# Expression in a Query

Each logged web requests includes the web page that was requested (the **stem**) and the query string (if any) passed in the request (**query**). Query strings can include parameters that are passed to the page in the name/value pair format *name*=*value*; with multiple parameters separated by a **&** character. For example, the web request *http://myserver/mypage.aspx?p1=12&p2=100* includes a stem (*/mypage.aspx*) and two parameter name/value pairs (*p1=12*, and *p2=100*).

In this procedure, you will use a complex C# expression to extract the parameters for each page request.

1. In the Azure portal, on the blade for your Azure Data Lake Analytics account, click **New Job**.
2. In the **New U-SQL Job** blade, in the **Job Name** box, type **Identify Parameters**.
3. In the code editor, enter the following code:

   ```
   USE DATABASE webdata;

   @paramstrings =
    SELECT stem AS page,
           new SQL.ARRAY<string>(
             query.Split('&').Where(x => x.Contains("="))
             ) AS paramstrings
   ```

```
FROM iis.log;

@params =
 SELECT DISTINCT page, param
 FROM @paramstrings
 CROSS APPLY EXPLODE(paramstrings) AS t(param);

OUTPUT @params
   TO "/output/params.csv"
   ORDER BY page, param
   USING Outputters.Csv();
```

This code queries the **iis.log** table to return a SQL.ARRAY object that contains an array of C# strings. The array is populated by splitting the query field into one or more strings based on the **&** delimiter, and then further filtering the parameters by using a C# lambda expression to include only values that include a **=** character.

Each array of parameter name/value pairs is then unpacked into the resultset by using the CROSS APPLY function with EXPLODE expression.

4. Click **Submit Job** and observe the job details as it is run.
5. When the job has finished, click the **Output** tab and select **params.csv** to see a preview of the results, which contains pages for which parameters were specified, and the parameter name/value pairs for each parameter requested.

Using a Code-Behind Class

To include custom functions in your U-SQL jobs, you can implement a code-behind class for your U-SQL query.

Note: To complete this exercise, you must be using a Windows computer with Visual Studio and the Visual Studio Azure Tools and SDK installed.

Create a U-SQL Visual Studio Project

The Visual Studio Azure Tools and SDK includes templates for Azure Data Lake projects.

1. Start Visual Studio.
2. Create a new project named **ProductCounts** based on the **U-SQL Project** template in the **Azure Data Lake** category. Save the project in the folder where you extracted the lab files.

Implement a Code-Behind Class File

As noted in the previous exercise, the web logs you have been working with in this course include the query strings that were passed in web requests. In this procedure, you will create a function to extract parameter values from these query strings. The custom function will be defined in a code-behind class file.

1. In Visual Studio, in the **Solution Explorer** pane, expand the **Script.usql** file to reveal the **Script.usql.cs** code-behind file.
2. Double-click **Script.usql.cs** to open it in the code editor pane.
3. Within the existing **ProductCounts** namespace, add the following code:

```
public static class Requests
{
  public static string GetParameter(string queryString,
                                    string parameterName)
  {
    string paramValue = "";
    int startParam = queryString.IndexOf(parameterName + "=");
    if (startParam >= 0)
    {
      int startVal = queryString.IndexOf("=", startParam) + 1;
      int endVal = queryString.IndexOf("&", startVal);
      if (endVal < 0)
      {
        endVal = queryString.Length;
      }
      paramValue = queryString.Substring(startVal, endVal - startVal);
    }
    return paramValue;
  }
}
```

This code defines a class named **Requests**, which contains a function named **GetParameter**. The **GetParameter** function attempts to find a specified parameter name in a provided query string, and if the parameter exists, the function extracts and returns its value.

4. Save the **Script.usql.cs** file.

Use the Custom Function in a U-SQL Query

Some of the query strings in the web logs include a **productid** parameter, indicating that the user viewed details about a specific product on your web site. In this exercise, you will implement a custom C# function in a code-behind file to parse the query string and extract a parameter value, enabling you to write a U-SQL query that counts the number of requests for each product.

1. In Solution Explorer, double-click **Script.usql** to view the (currently empty) U-SQL script file.
2. View the **Cloud Explorer** pane and if you are not already connected, sign into your Azure account in this pane. After you have signed in, you should be able to expand the **Data Lake Analytics** node under your subscription to see your Azure Data Lake Analytics service.
3. In the code editor pane, in the **Data Lake Analytics Accounts** drop-down list (in which **(local)** is currently selected), select your Azure Data Lake Analytics account. Then in the **Databases** list (in which **master** is currently selected), select **webdata**.
4. Add the following U-SQL code to the **Script.usql** code file:

```
@products =
    SELECT ProductCounts.Requests.GetParameter(query, "productid")
           AS product
    FROM iis.log
    WHERE query.Contains("productid");

@productRequests =
    SELECT product,
           COUNT(*) AS requests
    FROM @products
    GROUP BY product;

OUTPUT @productRequests
```

```
TO "/output/productRequests.csv"
USING Outputters.Csv();
```

This code uses the fully-qualified name of your function to extract the value of each **productid** parameter, and then uses that value in a query to aggregate the data by counting the number of requests for each **productid** value.

5. Save the **Script.usql** file.

Submit the Query

When you submit the query to your Azure Data Lake Analytics account from Visual Studio, the code-behind file is compiled into an assembly and temporarily deployed to the Azure Data Lake factory, where it can be used by your U-SQL query.

1. In the code editor pane, click **Submit**.
2. Observe the job details as it runs in the **Job View** pane.
3. When the job has finished, in the job graph, right-click the **productRequests.csv** output and click **Preview**.
4. When the preview opens, verify that the output includes a table of product IDs (for example *BC-R205*) and a count of requests for each product.
5. Return to the **Job View** pane, and under the job summary, click **Script** to view the script that was uploaded to Azure Data Lake Analytics.
6. Review the script, and note that Visual Studio automatically added CREATE ASSEMBLY and REFERENCE ASSEMBLY statements to the beginning of the script, and a DROP ASSEMBLY statement to the end of the script. These statements were used to enable your U-SQL code to access the custom function in your code-behind class, which was compiled into a .NET assembly and deployed to the Azure Data Lake catalog when you submitted the query.

Creating a Custom Assembly

To reuse custom functions in multiple U-SQL jobs, you can create a custom C# assembly and deploy it to the Azure Data Lake store.

Note: To complete this exercise, you must be using a Windows computer with Visual Studio and the Visual Studio Azure Tools and SDK installed.

Create a U-SQL Class Library

The Visual Studio Azure Tools and SDK includes a class library template for Azure Data Lake projects.

1. Start Visual Studio.
2. Create a new project named **DataUtilities** based on the **Class Library (for U-SQL Application)** template in the **Azure Data Lake** category. Save the project in the folder where you extracted the lab files.

Implement a Custom Class

A class library project includes only C# files, which you use to implement classes containing functions that you want to use from U-SQL scripts in Azure Data Lake Analytics. In this case, you will implement a simple utility class that includes a function to convert Bytes to Kilobytes.

1. In Visual Studio, in the **Solution Explorer** pane, open the **Class1.cs** code file if it is not already open.

2. Remove the existing using statements, and modify the remaining code as follows:

```
namespace DataUtilities
{
    public class Convertor
    {
        public static float BytesToKb(long? bytes)
        {
            return (float)bytes / 1000;
        }
    }
}
```

   This code defines a class named **Convertor**, which contains a function named **BytesToKb**. The **BytesToKb** function converts a specified value in Bytes to Kilobytes.

3. Save the **Class1.cs** file.

Deploy the Class Library

Before you can use the custom class library in a U-SQL query, you must register it in the Azure Data Lake Analytics account where you want to use it.

1. View the **Cloud Explorer** pane and if you are not already connected, sign into your Azure account in this pane. After you have signed in, you should be able to expand the **Data Lake Analytics** node under your subscription to see your Azure Data Lake Analytics service.
2. In the **Solution Explorer** pane, right-click the **DataUtilities** project and click **Register Assembly**.
3. In the **Assembly Registration** dialog box, in the **Analytics Account** drop-down list, select your Azure Data Lake Analytics account. Then in the **Database** drop-down list, select the **webdata** database.
4. Review the remaining settings in the dialog box, and then click **Submit**. The assembly will be compiled and deployed to your Azure Data Lake Analytics account.
5. When the class library has been registered, close Visual Studio.
6. In the Azure portal, on the blade for your Azure Data Lake Analytics account, click **Data Explorer**; and then browse to the **Assemblies** folder in your **webdata** database to verify that the assembly has been registered.

Use the Custom Class in a U-SQL Query

Now that you have registered the assembly containing your custom class, you can reference it in a U-SQL script.

1. In the Azure portal, on the blade for your Azure Data Lake Analytics account, click **New Job**.
2. In the **New U-SQL Job** blade, in the **Job Name** box, type **Use Custom Class**. Then change the **Parallelism** value to **4**.
3. In the code editor, enter the following code:

```
USE DATABASE webdata;

REFERENCE ASSEMBLY DataUtilities;

@kb =
SELECT date,
       DataUtilities.Convertor.BytesToKb(bytes_received) AS kb_received
```

```
 FROM iis.summary;

OUTPUT @kb
   TO "/output/kb.csv"
   ORDER BY date
   USING Outputters.Csv();
```

This code queries the **iis.summary** view, and uses the **DataUtilities.Convertor.BytesToKb** function you created in your custom class to convert the **bytes_received** value to Kilobytes.

4. Click **Submit Job** and observe the job details as it is run.
5. When the job has finished, click the **Output** tab and select **kb.csv** to see a preview of the results, which contains the number of Kilobytes received on each date.

Note: You will use the resources you created in this lab when performing the next lab, so do not delete them. Ensure that all jobs are stopped to minimize ongoing resource usage costs.