Microsoft

Processing Big Data with Hadoop in Azure HDInsight

Lab 3B – Using Python

Overview

In this lab, you will use Python to create custom user-defined functions (UDFs), and call them from Hive and Pig. Hive provides a powerful, but easy to use framework for working with data using SQL-like operations, and Pig offers an intuitive way to define a sequence of data transformations. In many scenarios, Hive and Pig provide all the functionality you need to transform and query your data. However, in some cases you might need to implement custom data processing logic that would be complex or impossible to achieve in Hive or Pig alone. Rather than resort to writing your own custom MapReduce components to achieve this; you can create a UDF that can be called from Hive or Pig. You can create UDFs in many languages, including Java and Microsoft C#; but increasingly Python is becoming the programming language of choice for Big Data processing.

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
- 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, and imported the publisher settings for your Azure subscription into PowerShell.

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.

Using a Python UDF from Hive

Hive supports Python UDFs through a *streaming* technique, in which data is passed between Hive and Python through the **stdin** and **stdout** interfaces. You can pass a line of data from Hive to Python through

stdin, unpack and process the data in Python, and then use the print command to return the results to Hive though **stdout**. By using this technique, you can write custom Python code to process your data, and then initiate data processing in a HiveQL script that passes the source data to the Python UDF for processing, receives the processed output, and returns it to the calling application.

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 the C:\HDILabs\Lab03B folder, rename **Provision HDInsight.txt** to **Provision HDInsight.ps1** (you may need to modify the *View* options for the folder to see file extensions). Then right-click **Provision HDInsight.ps1** and click **Run with PowerShell** to run the script (if you are prompted to change the execution policy, enter **Y**).
2. The script takes around 15-20 minutes to complete; so now is a really good time to go and make a cup of coffee! When the script has finished make a note of the details of your HDInsight cluster configuration. Then press ENTER to end the script.

Upload Files and Create Hive Tables

1. In the C:\HDILabs\Lab03B folder, rename **Upload Hive Files.txt** to **Upload Hive Files.ps1** (you may need to modify the *View* options for the folder to see file extensions). Then right-click **Upload Hive Files.ps1** and click **Edit** to open the script in the Windows PowerShell interactive script environment (ISE).
2. Change the value assigned to the **$clusterName** variable to match the name of your HDInsight cluster.

   Note: If you did not use the script provided to provision your cluster, you may also need to change the **$storageAccountName** and **$containerName** variables to match the names of your storage account and container.
3. Review the rest of the code in the script, and note that it performs the following actions:
   a. Uploads the **CreateHiveTables.txt** file to **/data** in your Azure storage account, and then starts a Hive job to run the HiveQL script that the file contains. This script creates a Hive table named **rawlogs** on the **/data/logs** folder.
   b. Uploads the files in the local **iislogs_gz** subfolder to **/data/logs** in your Azure storage container. These files are compressed IIS web server log files.
   c. Uploads the **convert_bytes.py** Python code file to **/data**.
4. Save the PowerShell script, and on the toolbar, click **Run Script**. Then wait for the script to finish and close Windows PowerShell ISE.
5. 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.
6. In the Azure portal, on the **HDInsight** page, select your HDInsight cluster and click **Query Console**. Then log into the query console using the HTTP user name and password for your cluster.
7. In the HDInsight query console, view the **Hive Editor** tab.
8. In the **Query Name** box, type **Query View**. Then replace the default **Select** statement with the following code (you can copy and paste this from **Query View.txt** in the C:\HDILabs\Lab04 folder):

```
set hive.execution.engine=tez;

SELECT * FROM vDailySummary;
```

9. Click **Submit** and wait for the job status in the **Job Session** table to change to **Running**. Then wait a few minutes until the status has changed to **Completed**, and click **Query View** in the **Query Name** column. This opens a new tab containing the job details.

10. In the **Job Details** tab, view the job output; noting that it summarizes the web server log data for each day to show the following fields:
    a. The date.
    b. The number of requests made to the web server.
    c. The amount of inbound data in bytes.
    d. The amount of outbound data in bytes.
11. Close the **Job Details** tab, but leave the **Hive Editor** tab open for the next task.

Use a Python UDF

1. Use Notepad to view the **convert_bytes.py** file in the C:\HDILabs\Lab03B folder.
2. Review the code this file contains, and note the following:
    a. Input data is read as a line from the **stdin** interface. When no more lines of data exist, the code exits a *while* loop and ends.
    b. The newline character (**\n**) is stripped from each line of input to create a row of data, and then the row is split into **log_date**, **requests**, **inbound_bytes**, and **outbound_bytes** variables based on a tab (**\t**) field terminator.
    c. A variable named **inbound_mbytes** is calculated as **inbound_bytes** multiplied by 1048576 (converting bytes to megabytes). Similarly, **outbound_mbytes** is calculated by converting **outbound_bytes** to megabytes.
    d. The **log_date**, **requests**, **inbound_mbytes**, and **outbound_mbytes** variables are joined to form a tab-delimited line, which is written to the **stdout** interface using the **print** command.
3. Close Notepad. Recall that this file was previously uploaded to your Azure storage container by a PowerShell script.
12. In your web browser, on the **Hive Editor** tab, in the **Query Name** box, type **Query UDF**. Then replace the existing code with the following code (you can copy and paste this from **Query UDF.txt** in the C:\HDILabs\Lab04 folder):

```
set hive.execution.engine=tez;

add file wasb:///data/convert_bytes.py;

SELECT TRANSFORM (log_date, requests, inbound_bytes, outbound_bytes)
  USING 'D:\Python27\python.exe convert_bytes.py' AS
  (log_date string, requests int, inbound_mbytes float, outbound_mbytes
float)
FROM vDailySummary;
```

13. Note that this code does the following:
    a. Enables the Tez engine.
    b. Adds the **convert_bytes.py** Python code file to the cluster cache (so that it is available to all cluster nodes).
    c. Uses the **TRANSFORM** HiveQL statement to pass the **log_date**, **requests**, **inbound_bytes**, and **outbound_bytes** values from **vDailySummary** to the **convert_bytes.py** Python script (specifying the path to the Python script engine executable).
    d. Defines a schema for the data returned by the Python script, including **log_date**, **requests**, **inbound_mbytes**, and **outbound_mbytes** fields.
14. Click **Submit** and wait for the job status in the Job Session table to change to **Running**. Then wait a few minutes until the status has changed to **Completed**, and click **Query UDF** in the **Query Name** column. This opens a new tab containing the job details.
15. In the **Job Details** tab, view the job output; noting that the Python UDF has returned the following web server log data for each day:

- The date.
- The number of requests made to the web server.
- The amount of inbound data in megabytes.
- The amount of outbound data in megabytes.

Using a Python UDF from Pig

Pig provides native support for Jython – a Java implementation of Python. This enables you to write Pig Latin code that calls Python UDFs directly, without having to use the streaming technique required by Hive.

View Source Data

1. Use Notepad to open the **ScrubbedWeather.txt** file in the C:\HDILabs\Lab03B folder.
2. Review text in this file. It consists of space-delimited rows containing values for **year**, **month**, **maxtemp**, **mintemp**, **frostdays**, **rainfall**, and **sunshinehours**.
3. When you have finished viewing the data, close Notepad. Do not save any changes.

View Python Code

1. Use Notepad to open the **convert_temp.py** file in the C:\HDILabs\Lab03B folder.
2. Review the Python code this file contains, and note that the code defines an output schema that includes a Pig *bag* structure named **f_readings**. The bag contains fields named **year**, **month**, **maxtemp**, **mintemp**, **frostdays**, **rainfall**, and **sunshinehours**. The code then defines a function named **fahrenheit** with an input parameter named **c_reading**. This function:
    a. Splits the input parameter into **year**, **month**, **maxtemp**, **mintemp**, **frostdays**, **rainfall**, and **sunshinehours** variables.
    b. Creates a variable named **maxtemp_f**, which is calculated as **maxtemp** multiplied by nine divided by five and added to 32 (the equation to convert Celsius to Fahrenheit).
    c. Similarly, creates a variable named **mintemp_f** with a value of **mintemp** converted to Fahrenheit.
    d. Returns the **year**, **month**, **maxtemp_f**, **mintemp_f**, **frostdays**, **rainfall**, and **sunshinehours** variables.
3. When you have finished viewing the code, close Notepad. Do not save any changes.

View Pig Latin Code

1. Use Notepad to open the **convert_weather.pig** file in the C:\HDILabs\Lab03B folder.
2. Review the Pig Latin code this file contains, and note that it performs the following tasks:
    a. Registers the **convert_temp.py** Python file as a Jython UDF.
    b. Loads the **scrubbedweather** source data into a relation named **Source**, with a single character array value for each line of text.
    c. Creates a relation named **ConvertedReadings** that uses the **fahrenheit** function in the **convert_temp.py** file to generate each row.
    d. Stores the **ConvertedReadings** relation in the **/data/convertedweather** folder.
3. When you have finished viewing the code, close Notepad. Do not save any changes.

Upload the Files and Run a Pig Job

1. In the C:\HDILabs\Lab03B folder, rename **Run Pig Script.txt** to **Run Pig Script.ps1** (you may need to modify the *View* options for the folder to see file extensions). Then right-click **Run Pig Script.ps1** and click **Edit** to open the script in the Windows PowerShell ISE.
2. Change the value assigned to the **$clusterName** variable to match the name of your HDInsight cluster.

Note: If you did not use the script provided to provision your cluster, you may also need to change the **$storageAccountName** and **$containerName** variables to match the names of your storage account and container.

3. Review the code in the script, noting that it performs the following tasks:
    a. Cleans up any leftover output from previous executions of this script.
    b. Uploads the **scrubbedweather.txt** source file to the **/data/scrubbedweather** folder in your Azure storage container.
    c. Uploads the **convert_temp.py** Python file to the **/data** folder in your Azure storage container.
    d. Uploads the **convert_weather.pig** Pig Latin script file to the **/data** folder in your Azure storage container.
    e. Starts an Azure HDInsight job to run the Pig script.
    f. Downloads the output from the Pig job and displays it in the console using the **cat** command.
4. Save the PowerShell script file and then click **Run Script** on the toolbar.
5. Wait for the script to finish and view the results that are displayed in the console pane. Then close Windows PowerShell ISE.

Cleaning Up

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

Note: If you are proceeding straight to the next lab, omit this task and use the same cluster in the next lab. Otherwise, follow the steps below to delete your cluster and storage account.

Delete the HDInsight Cluster

If you no longer need the HDInsight cluster used in this lab, you should delete it to avoid incurring unnecessary costs (or using credits in a free trial subscription). If you used the script provided in this lab to provision your cluster, you can use this procedure to delete it and its storage account automatically.

1. In the C:\HDILabs\Lab03B folder, rename **Delete HDInsight.txt** to **Delete HDInsight.ps1** (you may need to modify the *View* options for the folder to see file extensions). Then right-click **Delete HDInsight.ps1** and click **Run with PowerShell** to run the script (if you are prompted to change the execution policy, enter **Y**).
2. When prompted, enter the name of your HDInsight cluster.
3. When the script finishes (usually after a few minutes), press ENTER to end the script.