Analyzing Data with Spark in Azure Databricks

Lab 1 - Getting Started with Spark

Overview

In this lab, you will provision a Databricks workspace and a Spark cluster. You will then use the Spark cluster to explore data interactively.

What You'll Need

To complete the labs, you will need the following:

- A web browser
- A Microsoft account
- A Microsoft Azure subscription
- A Windows, Linux, or Mac OS X computer
- Azure Storage Explorer
- 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.

Provisioning Azure Resources

Note: If you already have an Azure Databricks Spark cluster and an Azure blob storage account, you can skip this section.

Provision a Databricks Workspace

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, click **+ Create a resource**. Then in the **Analytics** section select **Azure Databricks** and create a new Azure Databricks workspace with the following settings:
    - **Workspace 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 (and make a note of it!)*
    - **Location:** *Choose any available data center location.*
    - **Pricing Tier:** Standard
3. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the workspace to be deployed (this can take few minutes)

Provision a Storage Account

1. In the Azure portal tab in your browser, and click **+ Create a resource**.
2. In the **Storage** category, click **Storage account**.
3. Create a new storage account with the following settings:
   - **Name**: *Specify a unique name (and make a note of it)*
   - **Deployment model**: Resource manager
   - **Account kind**: Storage (general purpose v1)
   - **Location**: *Choose the same location as your Databricks workspace*
   - **Replication:** Locally-redundant storage (LRS)
   - **Performance:** Standard
   - **Secure transfer required:** Disabled
   - **Subscription:** *Choose your Azure subscription*
   - **Resource group:** *Choose the existing resource group for your Databricks workspace*
   - **Virtual networks:** Disabled
4. Wait for the resource to be deployed. Then view the newly deployed storage account.
5. In the blade for your storage account, click **Blobs**.
6. In the **Browse blobs** blade, click **+ Container**, and create a new container with the following settings:
   - **Name**: spark
   - **Public access level**: Private (no anonymous access)
7. In the **Settings** section of the blade for your blob store, click **Access keys** and note the **Storage account name** and **key1** values on this blade – you will need these in the next procedure.

Create a Spark Cluster

1. In the Azure portal, browse to the Databricks workspace you created earlier, and click **Launch Workspace** to open it in a new browser tab.
2. In the Azure Databricks workspace home page, under **New**, click **Cluster**.
3. In the **Create Cluster** page, create a new cluster with the following settings:
   - **Cluster Mode**: Standard
   - **Cluster Name**: *Enter a unique cluster name (and make a note of it)*
   - **Databricks Runtime Version**: *Choose the latest available version*
   - **Python Version:** 3
   - **Driver Type**: Same as worker
   - **Worker Type**: *Leave the default type*
   - **Min Workers:** 1
   - **Max Workers:** 2
   - **Auto Termination:** Terminate after 60 minutes.
   - **Spark Config**: Add two key-value pairs for your storage account and key like this:

     fs.azure.account.key.***your_storage_account***.blob.core.windows.net ***your_key1_value***

     spark.hadoop.fs.azure.account.key.***your_storage_account***.blob.core.windows.net ***your_key1_value***

Note: The first setting enables code using the newer Dataframe-based API to access your storage account. The second setting is used by the older RDD-based API.

4. Wait for the cluster to be created.

Exploring Data Interactively with Spark RDDs

Now that you have provisioned a Spark cluster, you can use it to analyze data. In this exercise, you will use Spark Resilient Distributed Datasets (RDDs) to load and explore data. The RDD-based API is an original component of Spark, and has largely been superseded by a newer Dataframe-based API; however, there are many production systems (and code examples on the Web) that use RDDs, so it's worth starting your exploration of Spark there.

Upload Source Data to Azure Storage

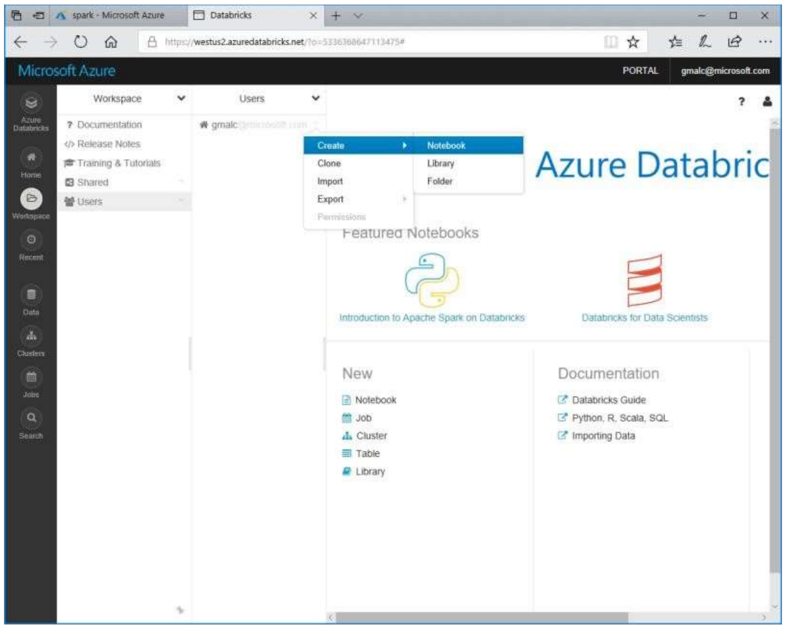
In this exercise, you will use the Spark RDD API to explore unstructured data. Before you can do this, you must store the data files you want to explore in a blob storage container where it can be accessed by your cluster. The instructions here assume you will use Azure Storage Explorer to do this, but you can use any Azure Storage tool you prefer.

1. In the folder where you extracted the lab files for this course on your local computer, in the **data** folder, verify that the **KennedyInaugural.txt** file exist. This file contains the data you will explore in this exercise.
2. Start Azure Storage Explorer, and if you are not already signed in, sign into your Azure subscription.
3. Expand your storage account and the **Blob Containers** folder, and then double-click the **spark** blob container you created previously.
4. In the **Upload** drop-down list, click **Upload Files**. Then upload **KennedyInaugural.txt** as a block blob to a new folder named **data** in root of the **spark** container.

Create a Notebook

Most interactive data analysis in Databricks is conducted using *notebooks*. These browser-based interactive documents enable you to combine notes in Markdown format with code that you can run right in the notebook – with no requirement to install a local code editing environment. In this exercise, you can choose to write your code using Python or Scala.

1. In the Databricks workspace, click **Workspace**. Then click **Users**, click your user name, and in the drop-down menu for your username point click **Create** and **Notebook** as shown here:



2. Create a new notebook with the following settings: • **Name**: RDDs
   - **Language**: *Choose* Python *or* Scala *as preferred*.
   - **Cluster**: *Your cluster*

3. In the new notebook, in the first cell, enter the following code to enter some Markdown text:

```
%md
# Kennedy Inauguration
This notebook contains code to analyze President Kennedy's inauguration speech.
```

4. Click anywhere in the notebook outside of the first cell to see the formatted markdown, which should look like this:

Kennedy Inauguration

This notebook contains code to analyze President Kennedy's inauguration speech.

5. Hold the mouse pointer under the center of the bottom edge of the cell until a **(+)** symbol is displayed; then click this to insert a new cell.
6. In the new cell, type the following code, replacing ***<account>*** with the fully qualified name of your Azure Storage account (***account_name*.blob.core.windows.net**):

Python

```
txt = sc.textFile("wasbs://spark@<account>/data/KennedyInaugural.txt")
txt.count()
```

Scala

```
val txt = sc.textFile("wasbs://spark@<account>/data/KennedyInaugural.txt")
txt.count()
```

In this code, the variable **sc** is the Spark context for your cluster; which is created automatically within the notebook.

7. With the code cell selected, at the top left of the cell, click the ▶▾ button and then click ▶ **Run Cell** to run the cell. After a few seconds, the code will run and display the number of lines of text in the text file as **Out[1]**
8. Add a new cell and enter the following command to view the first line in the text file.

   Python `txt.first()`

   Scala `txt.first()`

9. Run the new cell and note that the first line of the speech is displayed as **Out[2]**.
10. Add a new cell and enter the following command to create a new RDD named **filtTxt** that filters the **txt** RDD so that only lines containing the word "freedom" are included, and counts the filtered lines.

    Python
    ```
    filtTxt = txt.filter(lambda line: "freedom" in line) filtTxt.count()
    ```

    Scala
    ```
    val filtTxt = txt.filter(line => line.contains("freedom")) filtTxt.count()
    ```

11. Run the new cell and note that the number of lines containing "freedom" is returned as **Out[3]**.
12. Add a new cell and enter the following command to display the contents of the **filtTxt** RDD.

    Python `filtTxt.collect()`

    Scala `filtTxt.collect()`

13. Run the new cell and note that the lines containing "freedom" are returned as **Out[4]**.
14. Add a new cell and enter the following command to split the full speech into words, count the number of times each word occurs, and display the counted words in descending order of frequency.

    Python
    ```
    words = txt.flatMap(lambda txt: txt.split(" "))
    counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)
    counts.sortBy(lambda a: a[1], False).collect()
    ```

    Scala
    ```
    val words = txt.flatMap(line => line.split(" "))
    val counts = words.map(word => (word, 1)).reduceByKey((a, b) => a + b)
    counts.sortBy(_._2,false).collect().foreach(println)
    ```

15. Run the new cell and review the output, which shows the frequency of each word in the speech in descending order.

Exploring Data Interactively with Dataframes

Spark 2.0 and later provides a schematized object for manipulating and querying data – the DataFrame. This provides a much more intuitive, and better performing, API for working with structured data. In addition to the native Dataframe API, Spark SQL enables you to use SQL semantics to create and query tables based on Dataframes.

Upload Source Data to Azure Storage

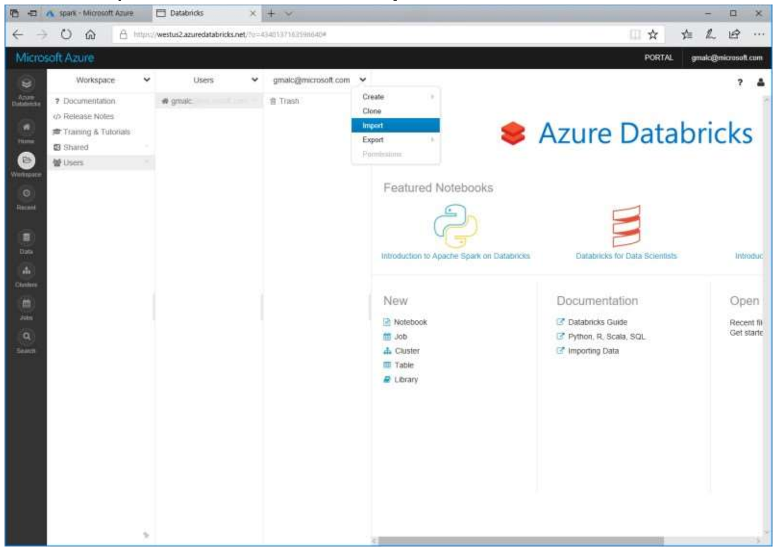
In this lab, you will explore both structured data relating to road traffic accidents. Before you can do this, you must upload the data files containing the data to your blob storage container where it can be accessed by your cluster. The instructions here assume you will use Azure Storage Explorer to do this, but you can use any Azure Storage tool you prefer.

1. In the folder where you extracted the lab files for this course on your local computer, in the **data** folder, verify that the **Accidents.csv** and **Vehicles.csv** files exist. These files contain the data you will explore in this exercise.
2. Start Azure Storage Explorer, and if you are not already signed in, sign into your Azure subscription.
3. Expand your storage account and the **Blob Containers** folder, and then double-click the **spark** blob container you created previously in this lab.
4. In the **Upload** drop-down list, click **Upload Files**. Then upload **Accidents.csv** and **Vehicles.csv** as block blobs to the **data** folder in root of the **spark** container that you created previously in this lab.

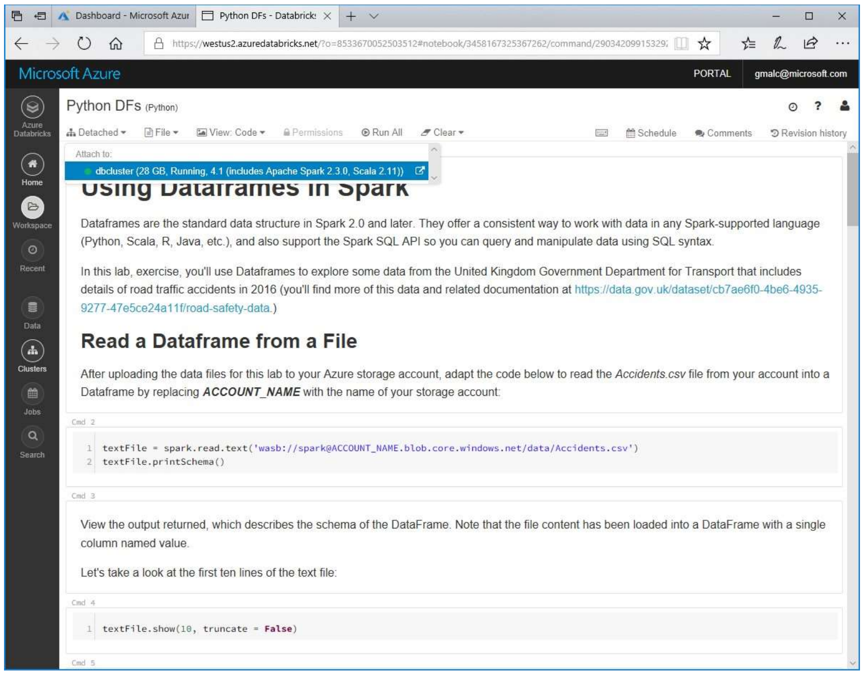
Work with Dataframes

In this procedure, you will use your choice of Python or Scala to query the road traffic accident data in the comma-delimited text files you have uploaded. Notebooks containing the necessary steps to explore the data have been provided.

1. In the Databricks workspace, click **Workspace**. Then click **Users**, click your user name, and in the drop-down menu for your username click **Import** as shown here:



2. Browse to the folder where you extracted the lab files. Then select either **Dataframes.ipynb** or **Dataframes.scala**, depending on your preferred choice of language (Python or Scala), and upload it.
3. Open the notebook you uploaded and in the **Detached** drop-down menu, attach the notebook to your Spark cluster as shown here:



Using Dataframes in Spark

Dataframes are the standard data structure in Spark 2.0 and later. They offer a consistent way to work with data in any Spark-supported language (Python, Scala, R, Java, etc.), and also support the Spark SQL API so you can query and manipulate data using SQL syntax.

In this lab, exercise, you'll use Dataframes to explore some data from the United Kingdom Government Department for Transport that includes details of road traffic accidents in 2016 (you'll find more of this data and related documentation at https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data.)

Read a Dataframe from a File

After uploading the data files for this lab to your Azure storage account, adapt the code below to read the *Accidents.csv* file from your account into a Dataframe by replacing ***ACCOUNT_NAME*** with the name of your storage account:

```
textFile = spark.read.text('wasb://spark@ACCOUNT_NAME.blob.core.windows.net/data/Accidents.csv')
textFile.printSchema()
```

View the output returned, which describes the schema of the DataFrame. Note that the file content has been loaded into a DataFrame with a single column named value.

Let's take a look at the first ten lines of the text file:

```
textFile.show(10, truncate = False)
```

4. Read the notes and run the code cells to explore the data.

Clean Up

Note: If you intend to proceed straight to the next lab, skip this section. Otherwise, follow the steps below to delete your Azure resources and avoid being charged for them when you are not using them.

Delete the Resource Group

1. Close the browser tab containing the databricks workspace if it is open.
2. In the Azure portal, view your **Resource groups** and select the resource group you created for your databricks workspace. This resource group contains your databricks workspace and your storage account.
3. In the blade for your resource group, click **Delete**. When prompted to confirm the deletion, enter the resource group name and click **Delete**.
4. Wait for a notification that your resource group has been deleted.
5. After a few minutes, a second resource group containing the resources for your cluster will automatically be deleted.
6. Close the browser.