Lab: Getting started with Spark on Azure HDInsight

Contents

Overview

Summary

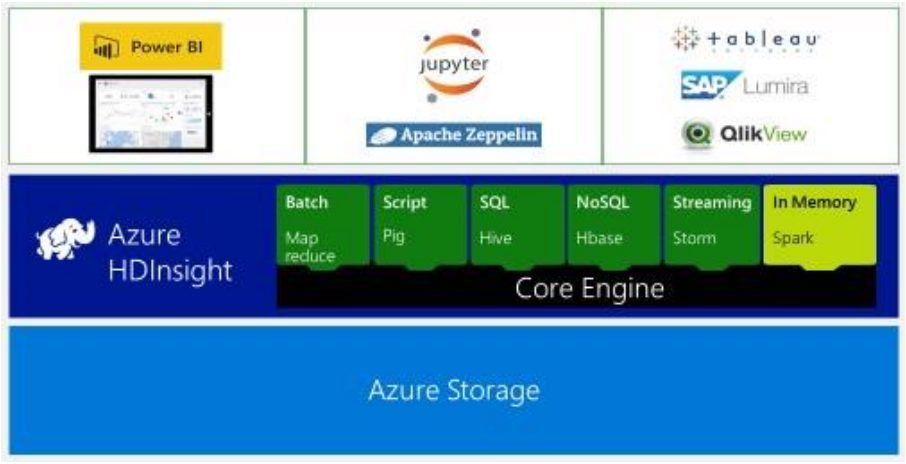
This lab is intended to serve as a general introduction to Spark on HDInsight and will take about 60 minutes to complete. We will be covering:

- Provisioning an Apache Spark cluster on HDInsight
- Using web-based notebooks Zeppelin (Scala/Spark SQL) and Jupyter (Python) to run Spark SQL queries on the cluster

Apache Spark is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Spark's in-memory computation capabilities make it a good choice for iterative algorithms in machine learning and graph computations.

Spark can also be used to perform conventional disk-based data processing. Spark improves the traditional MapReduce framework by avoiding writes to disk in the intermediate stages. Also, Spark is compatible with the Hadoop Distributed File System (HDFS) and Azure Blob storage so the existing data can easily be processed via Spark.

To interact with Spark there are several options including Power BI, Jupyter, Zeppelin and 3rd party BI tools (see graphic below).


Power BI
jupyter
Apache Zeppelin
tableau
SAP Lumira
QlikView
Azure HDInsight
Batch
Map reduce
Script
Pig
SQL
Hive
NoSQL
Hbase
Streaming
Storm
In Memory
Spark
Core Engine
Azure Storage

What is Apache Zeppelin?

Apache Zeppelin is a web-based notebook that enables interactive data analytics. The Zeppelin interpreter allows any language/data-processing-backend to be plugged in. However, current languages included in the Zeppelin interpreter are:

- Scala (with Apache Spark)
- SparkSQL
- Markdown
- Shell

Zeppelin notebooks hosted on the HDInsight Spark Cluster supports Scala and SparkSQL. In this lab will we not be using Zeppelin. Instead we will focus on Jupyter.

Please take a moment to read more details on the Apache Zeppelin website.

What is Jupyter?

Jupyter is another web-based notebook that enables interactive data analytics. Original Jupyter got its name because it was meant as a notebook frontend for the Ju(lia)Pyt(hone)R programming languages. Today, Jupyter supports over 40 programming languages.

Jupyter notebooks on the HDInsight Spark cluster support Python. In this lab we will leverage the pyspark package to run Spark SQL queries.

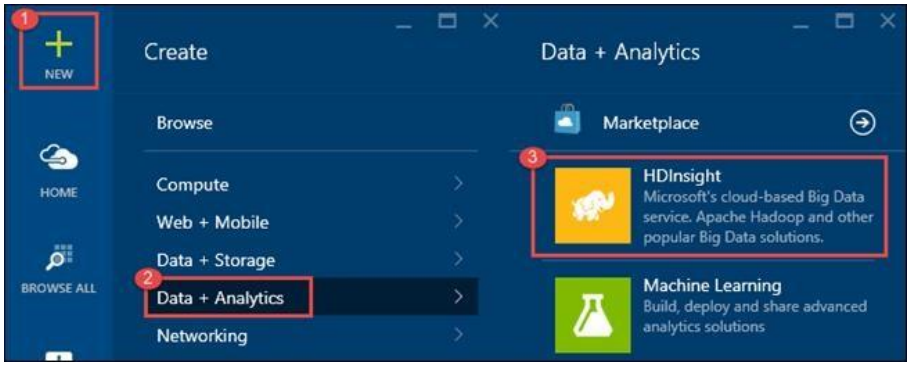
Please take a moment to read more details on the Jupyter website.

Lab Requirements/Prerequisites

Before you begin this tutorial, you must have an Azure subscription. See Get Azure free trial.

Step 1: Provision an HDInsight Spark Cluster

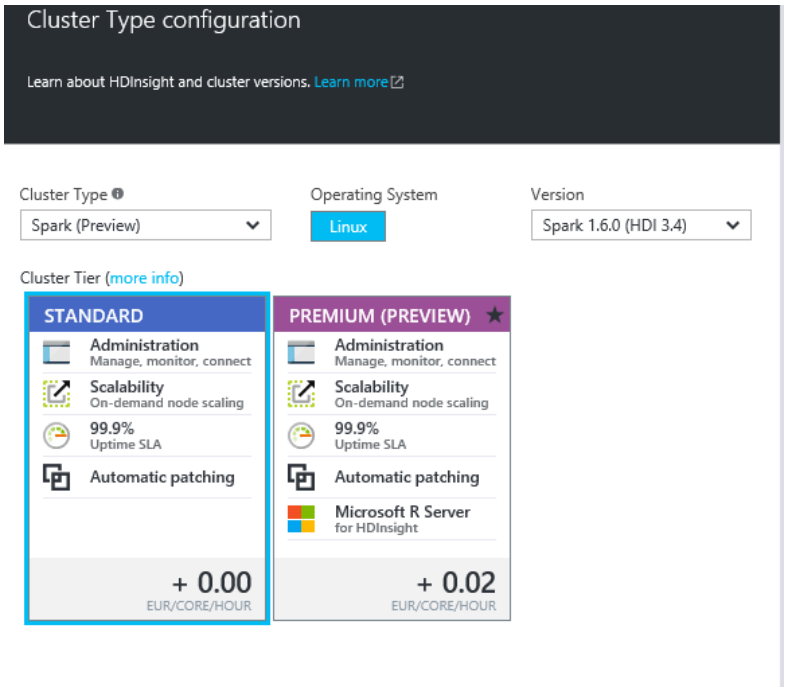
1. Sign in to the Azure preview portal.
2. Click **NEW**, click **Data + Analytics**, and then click **HDInsight**.



3. Enter a **Cluster Name**. A green check appears beside the cluster name if it is available.

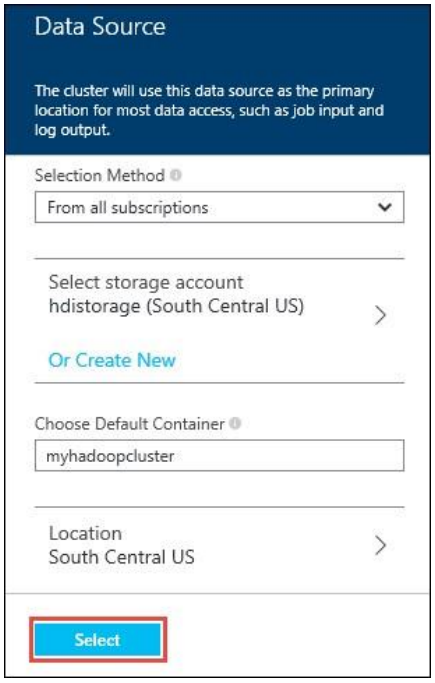
4. If you have more than one subscription, click the **Subscription** entry to select the Azure subscription to use for the cluster.

5. Click **Select Cluster Type** and select **Spark (Preview)** as the cluster type. Select the **Standard** cluster Tier (see below).



6. Click **Credentials**, and then enter a **Cluster Login Username**, a **Cluster Login Password**, the **SSH Username** and **SSH Password**. Click **Select** to save the settings.

7. Click **Data Source** to choose an existing data source for the cluster, or **create a new one**. When you provision a Hadoop cluster in HDInsight, you specify an Azure Storage account. A specific Blob storage container from that account is designated as the default file system, like in the Hadoop distributed file system (HDFS). By default, the HDInsight cluster is provisioned in the same data center as the storage account you specify. For more information, see Use Azure Blob storage with HDInsight.



Currently you can select an Azure Storage Account as the data source for an HDInsight cluster. Use the following to understand the entries on the Data Source blade.

- **Selection Method**: Set this to **From all subscriptions** to enable browsing of storage accounts from all your subscriptions. Set this to **Access Key** if you want to enter the **Storage Name** and **Access Key** of an existing storage account.

- **Select storage account / Create New**: Click **Select storage account** to browse and select an existing storage account you want to associate with the cluster. Or, click **Create New** to create a new storage account. Use the field that appears to enter the name of the storage account. A green check appears if the name is available.

- **Choose Default Container**: Use this to enter the name of the default container to use for the cluster. While you can enter any name here, we recommend using the same name as the cluster so that you can easily recognize that the container is used for this specific cluster.

- **Location**: The geographic region that the storage account is in, or will be created in.

IMPORTANT: Selecting the location for the default data source also sets the location of the HDInsight cluster. The cluster and default data source must be located in the same region.

8. Click **Node Pricing Tiers** and accept the defaults. By default, the size of the cluster is 4 worker nodes and 2 head nodes. The estimated cost of this cluster is approximately 80pence per hour per node. You can reduce the size of the cluster to 2 worker nodes (head node size cannot be reduced) without affecting the ability to run this lab.

9. Click **Resource Group** to see a list of existing resource groups and select where to create the cluster. Or, you can click **Create New** and then enter the name of the new resource group. A green check appears to indicate if the new group name is available.

10. On the **New HDInsight Cluster** blade, ensure that Pin to Startboard is selected, and then click Create. This creates the cluster and adds a tile for it to the Startboard of your Azure portal. The icon will indicate that the cluster is provisioning, and will change to display the HDInsight icon once provisioning has completed.

> **NOTE:** It will take some time for the cluster to be created, usually around 15 minutes. Use the tile on the Startboard, or the **Notifications** entry on the left of the page to check on the provisioning process.

11. Once the provisioning is complete, click the tile for the Spark cluster from the Startboard to launch the cluster blade.

Step 2: Run interactive Spark SQL queries using Jupyter

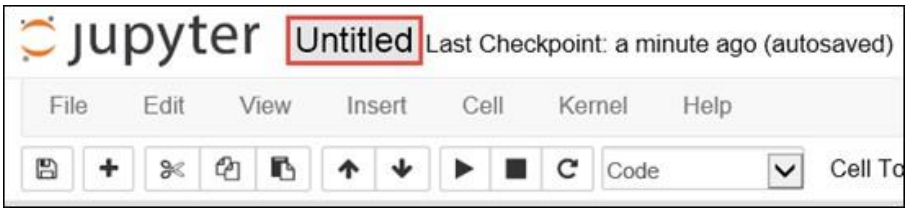
1. From the Azure Preview Portal, from the startboard, click the tile for your Spark cluster (if you pinned it to the startboard). You can also navigate to your cluster under **Browse All > HDInsight Clusters**.

2. From the Spark cluster blade, click **Quick Links**, and then from the **Cluster Dashboard** blade, click **Jupyter Notebook**. If prompted, enter the admin credentials for the cluster.

> **NOTE:** You may also reach the Jupyter Notebook for your cluster by opening the following URL in your browser.
> Replace **CLUSTERNAME** with the name of your cluster:
>
> https://CLUSTERNAME.azurehdinsight.net/jupyter

3. Create a new notebook. Click in the top right **New**, and then click **PySpark**. Do not worry if you get a new screen with a 404, just close the window and continue to the next step.

4. A new notebook is created and opened with the name Untitled.pynb. Select the notebook to open it. Click the notebook name at the top, and enter a friendly name.



5. Import the required modules and create the Spark and SQL contexts. Write the following code example in an empty cell, and then press **SHIFT + ENTER**. Note that the code snippets are also available in a separate TXT file.

```
from pyspark.sql.types import *
```

On the first run, the Spark context is established. On subsequent runs, we don't need to repeat that. If you do run this command multiple times you may receive the following error:

```
Error
-----------------------------------------------------------
ValueError Traceback (most recent call last)
<ipython-input-2-ad7e7ec9ed75> in <module>()
4
5 # Create Spark and SQL contexts
----> 6 sc = SparkContext('spark://headnodehost:7077',
'pyspark')
7 sqlContext = SQLContext(sc)
.
.
.

ValueError: Cannot run multiple SparkContexts at once; existing
SparkContext(app=pyspark,
master=spark://headnodehost:7077) created by __init__ at
<ipython-input-1-ad7e7ec9ed75>:6
```

Every time you run a job in Jupyter, your web browser window title will show a (**Busy**) status along with the notebook title. You will also see a solid circle next to the **PySpark** text in the top-right corner. After the job is completed, this will change to a empty circle.

6. Load sample data into a temporary table. When you provision a Spark cluster in HDInsight, the sample data file, **hvac.csv**, is copied to the associated storage account under **\HdiSamples\SensorSampleData\hvac**.

In an empty cell, paste the following code example and press **SHIFT + ENTER**. This code example registers the data into a temporary table called **hvac**.

```
# Load the data
hvacText =
sc.textFile("wasb:///HdiSamples/HdiSamples/SensorSampleData/hvac/HVAC.csv")

# Create the schema
hvacSchema = StructType([StructField("date", StringType(),
False),StructField("time", StringType(), False),StructField("targettemp",
IntegerType(), False),StructField("actualtemp", IntegerType(),
False),StructField("buildingID", StringType(), False)])

# Parse the data in hvacText
hvac = hvacText.map(lambda s: s.split(",")).filter(lambda s: s[0] !=
"Date").map(lambda s:(str(s[0]), str(s[1]), int(s[2]), int(s[3]), str(s[6])
))

# Create a data frame
hvacdf = sqlContext.createDataFrame(hvac,hvacSchema)

# Register the data frame as a table to run queries against
hvacdf.registerTempTable("hvac")
```

7. Because you are using a PySpark kernel, you can now directly run a SQL query on the temporary table **hvac** that you just created by using the `%%sql` magic. For more information about the `%%sql` magic, as well as other magics available with the PySpark kernel, see [Kernels available on Jupyter notebooks with Spark HDInsight clusters].

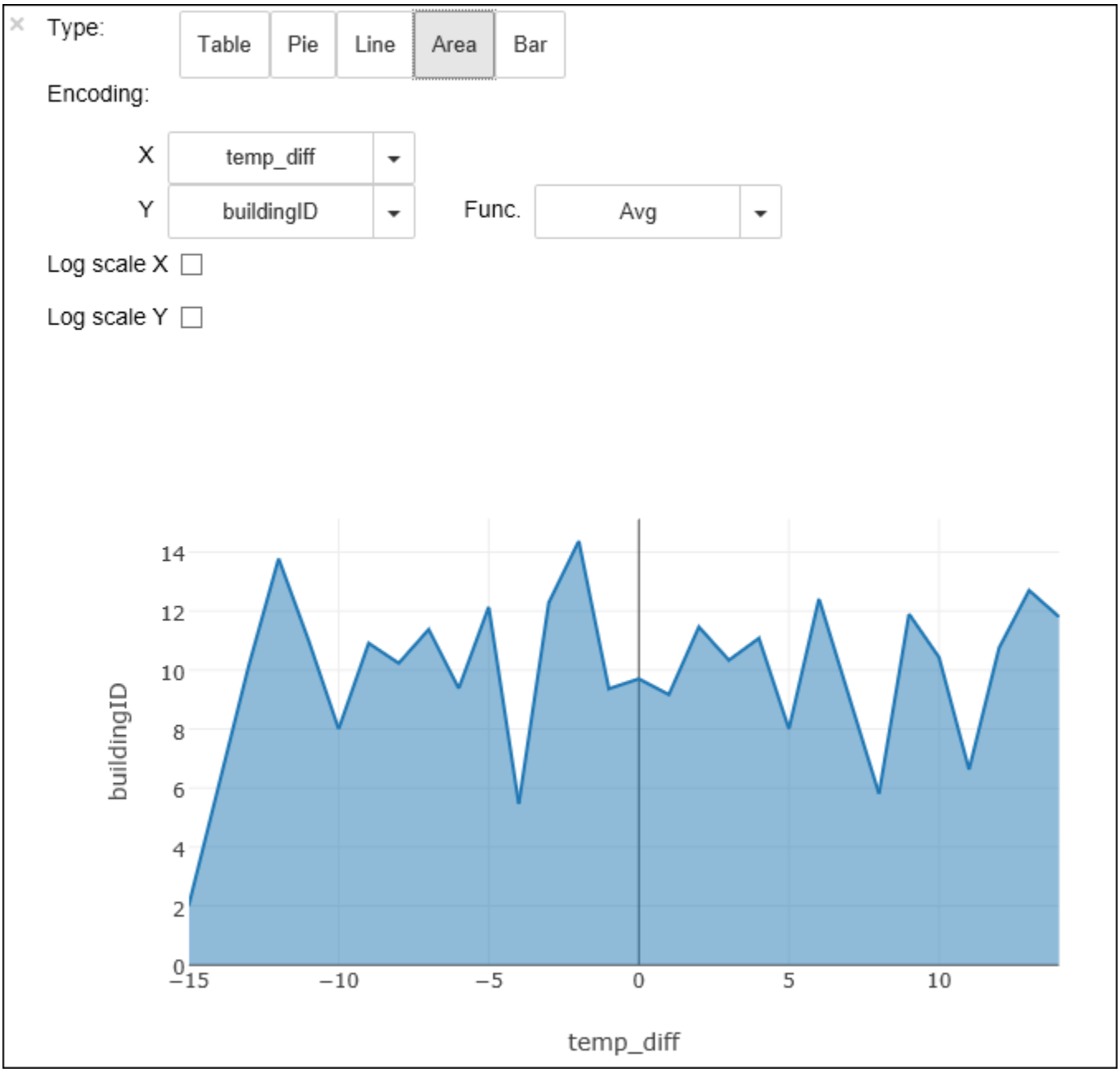
   ```
   %%sql
   SELECT buildingID, (targettemp - actualtemp) AS temp_diff, date FROM hvac
   WHERE date = \"6/1/13\"
   ```

8. Once the job is completed successfully, the following output is displayed.

```
buildingID temp_diff date
4          8         6/1/13
3          2         6/1/13
7          -10       6/1/13
12         3         6/1/13
7          9         6/1/13
7          5         6/1/13
3          11        6/1/13
8          -7        6/1/13
17         14        6/1/13
16         -3        6/1/13
8          -8        6/1/13
1          -1        6/1/13
12         11        6/1/13
3          14        6/1/13
6          -4        6/1/13
1          4         6/1/13
19         4         6/1/13
19         12        6/1/13
9          -9        6/1/13
15         -10       6/1/13
```

```
In [ ]:
```

9. You can also see the results in other visualizations as well. For example, an area graph for the same output would look like the following.



10. After you have finished running the application, you should shutdown the notebook to release the resources. To do so, from the **File** menu on the notebook, click **Close and Halt**. This will shutdown and close the notebook.