Microsoft

Exploring Apache Spark and Spark SQL in Microsoft Azure HDInsight

Change management

Version	Date Effective	Incorporated Changes	Requested By
1.0	06/01/2016	Initial Version	Nishant Thacker
1.1	07/12/2016	Formatting changes	Nishant Thacker
1.2	01/29/2017	TechReady 24	Nishant Thacker
1.3	07/17/2017	Ready 2017	Nishant Thacker

Contents

Introduction

This class introduces students to Apache Spark on Azure HDInsight. It helps student to understand the value proposition of Apache Spark over other Big Data technologies like Hadoop. They should understand the similarities between Hadoop & Spark, their differences and respective nuances. They should be able to decide when to use what and why for a given business use case in a typical enterprise environment.

Azure specific highlights of Apache Spark

Source: http://azure.microsoft.com/en-us/services/hdinsight/apache-spark/

#1 Ease of Deployment of Spark over the Azure Cloud

Users can create Spark clusters with HDInsight in minutes.

#2 Every cluster comes with Jupyter notebook for interactive data analysis

Every Spark for Azure HDInsight cluster has Jupyter notebook included. This allows users to do interactive and exploratory analysis in Scala, Python or SQL.

#3 Scale-up or Scale-down a running Spark cluster as per business needs

Users can take advantage of the elasticity of the cloud by using the Scale feature on every HDInsight Spark cluster using which they can scale-up or scale-down a running Apache Spark cluster.

Spark SQL is Spark's module for working with structured data

This class specifically focuses on Spark SQL module and highlights its value proposition in the Apache Spark Big Data processing framework.

Main highlights of Spark SQL

Source: http://spark.apache.org/sql/

#1 Integrated - Seamlessly mix SQL queries with Spark programs.Spark SQL lets users query structured data inside Spark programs, using either SQL or a familiar DataFrame API. Usable in Java, Scala, Python and R.

#2 Uniform Data Access - Connect to any data source the same way.DataFrames and SQL provide a common way to access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC. Users can even join data across these sources.

#3 Hive Compatibility - Run unmodified Hive queries on existing data.Spark SQL reuses the Hive frontend and metastore, giving users full compatibility with existing Hive data, queries, and UDFs.

#4 Standard Connectivity - Connect through JDBC or ODBC.A server mode provides industry standard JDBC and ODBC connectivity for business intelligence tools.

Takeaways

Provision an HDInsight Spark Cluster.

Access data from Azure storage container and create Dataframe.

Understand joins, functions and user defined functions.

Prerequisites

a) An Azure subscription. See Get Azure free trial.

b) Download and install Azure Storage Explorer.

c) Power BI account.

Ready 2017 special instructions

HDInsight cluster usually take 15-20 minutes to create. As a work around to the cluster create time, we've pre-provisioned HDI clusters for this lab and are sharing the credentials below. Note that these clusters are only active for TechReady and will be deleted after the event, so if you're trying the labs after TR, you should create your own clusters and use the cluster properties to proceed with the lab.

These credentials are shared in good faith and the understanding is that attendees will not misuse these for any purposes, including but not limited to this lab. If you have any concerns, please close this lab now and do not proceed any further.

Ready 2017 Cluster Credentials:

*Note: The steps in the following section '*Provision HDInsight Linux Hadoop cluster with Azure Management Portal *' should be ignored if you are provided a shared cluster. For TechReady you're provided a cluster with the following credentials:*

Spark Cluster:

Cluster Name for Spark Cluster: nthdilabsspark

Cluster URL (Ambari) for Hive Cluster: https://nthdilabsspark.azurehdinsight.net/

Username: admin

Password: HDItut@123

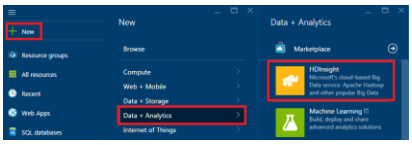
ClassSection 1: Provision an HDInsight Spark cluster

Access Azure Preview Portal

1. Sign in to the Azure preview portal.

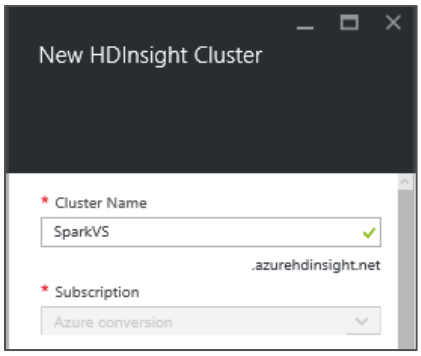
Create HDInsight Spark cluster

1. Click **NEW**, click **Data + Analytics**, and then click **HDInsight**.



Provide Cluster Details

1. In the New HDInsight Cluster blade , enter **Cluster Name.**

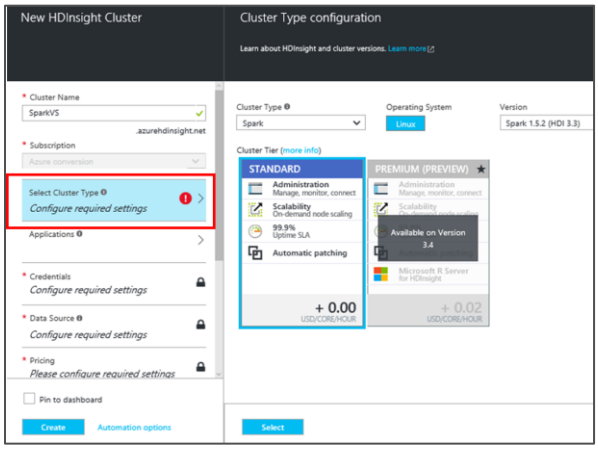


A green check mark appears beside the cluster name if it is available.

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

Configure Cluster Type

1. Click on **Select Cluster type.** This will open the Cluster Type configuration blade.

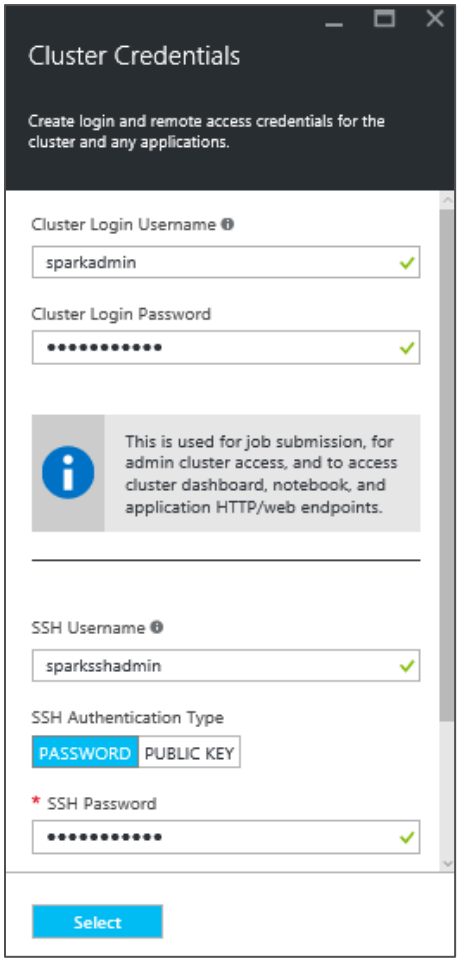


2. Select **Spark** as **Cluster Type.**

3. **Operating System** will be **Linux** by default.
4. Select **Version** as **Spark 1.6.2 (HDI 3.4)**
5. For **Cluster Tier**, select **STANDARD**
6. Click **SELECT** to complete the configuration settings

Provide credentials to access cluster

1. Click **Credentials** tab to open **Cluster Credentials blade.**



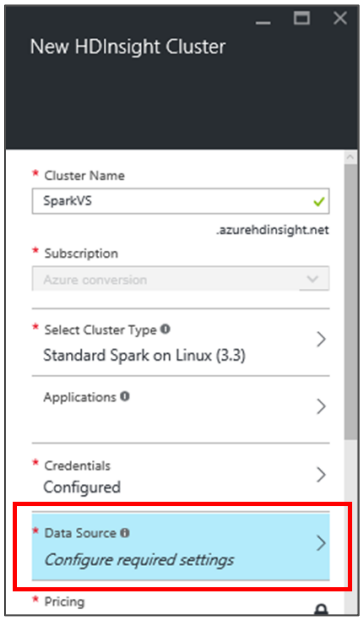
2. Enter **Cluster Login Username.**
3. Enter **Cluster Login Password**.
4. Enter **SSH Username**.
5. Select **PASSWORD** as **SSH Authentication Type**.
6. Enter **SSH Password** and confirm it.
7. Click **Select** button to save the credentials.

**SSH Username and Password is required to remote session of Spark Cluster*

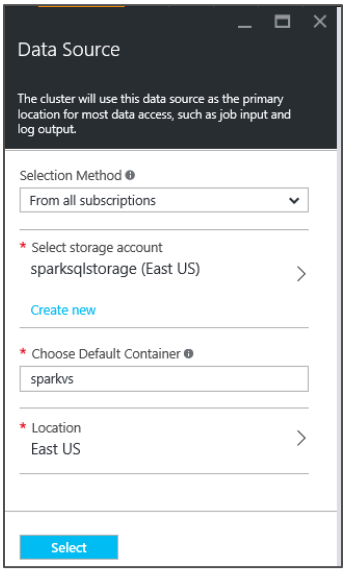
Provide Data Source

Data source will be used as primary location for most data access, such as job input and log output.

1. Click **Data Source tab** present on New HDInsight Cluster blade



2. In the Data source blade**, Selection Method** provides two options:



Option 1 - Set this to **Access Key** if you want to use existing storage account and you have **Storage Name** and **Access Key** of same, else

Option 2 - select **From all subscriptions** as Selection method.

Select **From all subscriptions** for purpose of this Lab exercise

3. To create new storage account enter a name for new storage account in **Create New Storage Account** input box

   Or
   Click on link **Select Existing** to select from existing accounts.
   For purspose of this exercise, we will create a new storage account.

4. Enter name for default container to be designated for cluster in **Choose Default Container** field**.**

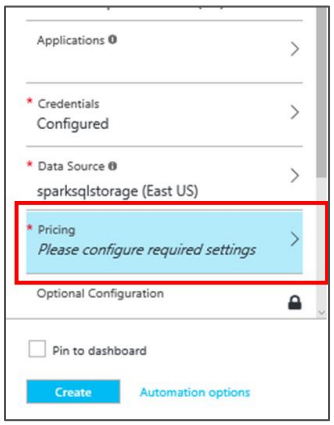
5. If existing storage account is selected then no need to provide **Location** else select appropriate **Location**.

   By default, the HDInsight cluster is provisioned in the same data center as the storage account you specify

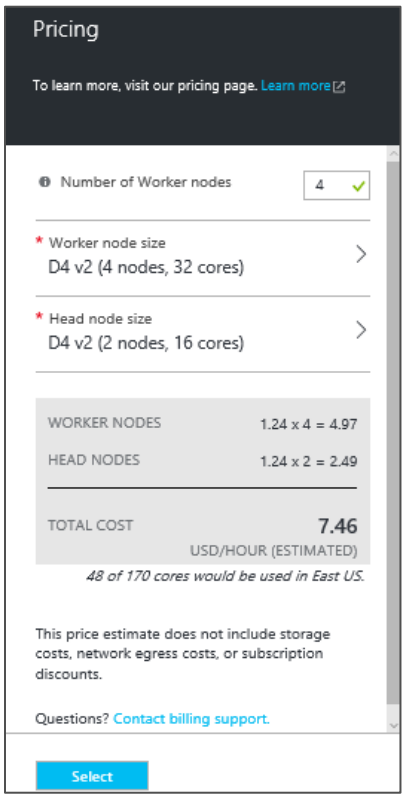
6. Click Select button at the bottom to save the data source configuration.

Set Pricing

1. Click **Pricing** tab to open the Pricing blade.



2. The Pricing blade provides options to the configure number of nodes in cluster, which will be the base pricing criteria. Enter number of worker node in **Number of Worker nodes** field, set it to **4** for this demo.



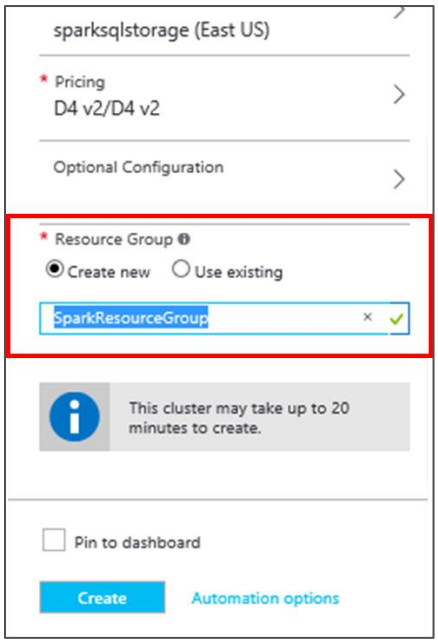
3. Leave all other values as default.

   Note that based on the number of worker notes and size, the estimated cost of the cluster is calculated and displayed in USD/HOUR.

4. Click **Select** button to save node pricing configuration.

Provide Resource Group

1. In the **Resource Group** section, you have options:



   a) Click link Create New to provide new Resource Group.

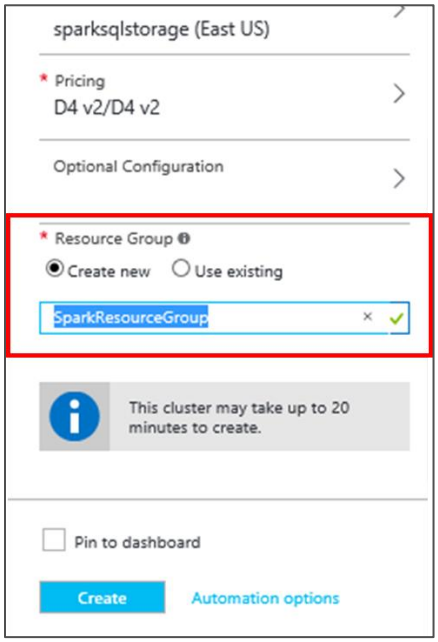
      Or

   b) Use Existing by selecting appropriate Resource Group from list.

   For the purpose of this lab, we will create a new resource group.

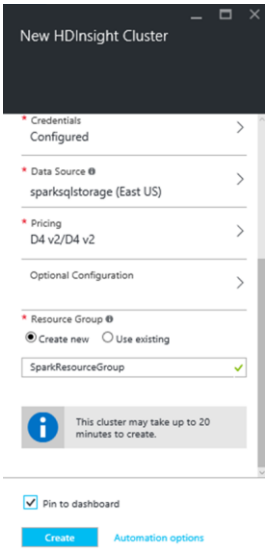
2. Enter name for Resource Group

   A green check appears beside the cluster name if this name is available.



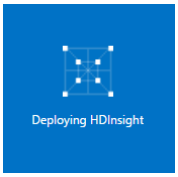
Provision cluster

1. After completing all the configraution, in the New HDInsight Cluster blade, make sure to tick on the 'Pin to dashboard' option.
2. Click **Create** button to finalize cluster creation.



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.



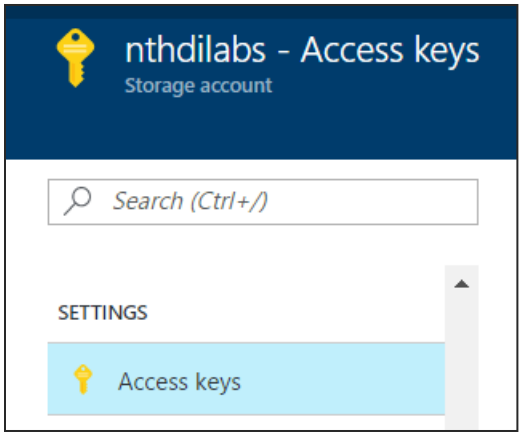
Section 2: Load datasets files to storage account.

In this section, you'll copy the files required for the lab to your storage account. You'll copy the files between two storage acount with the help of AzCopy utility. You can download the utility from here http://aka.ms/downloadazcopy

Note: Ignore Step 1 and 2 if you are provided a shared cluster (for Ready)

To copy the files, follow the below steps.

1. Copy your Azure Storage account access keys. This is required to copy data from the source Azure Storage account to your Azure Storage account. To get your storage account access key, navigate to your storage account on the Azure Management Portal and select **Access keys** under **Settings**.



2. Click on the copy icon to copy **Key1** from the **Access Keys** pane.

3. Press Window + R to open the run window. Type cmd and press enter to open a new command console window.

4. Change the directory to C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy.

5. Copy and paste the following command on the console window to transfer **all spark lab assets needed** from the source storage account to your storage account.

```
AzCopy /Source:https://nthdilab.blob.core.windows.net/sparklabs/
/Dest:https://nthdilabs.blob.core.windows.net/nthdilabsspark/<yourname>/SparkLab
Datasets/
/SourceKey:G1t6T13jn3K6w/4ZS2NG7RsTHe5YuusRLd9tKnRlJku7cjCwcRk5JxWAHmtrH1Dt03+nw
ttYbB2HuvHeI/UiNw==
/DestKey:QfPYhbGxokkcqjjEZIkbzzyDdMb7QHSrUjA6UnpfuLjC0Np4PSSjkfrsnVhGyOxJsKWEK9C
XNvPZo/L1w7T0ow== /S
```

Note: Replace <yourname> with your name or a unique ID, so that your files do not get processed by other jobs.

Section 3: Access data from Azure storage container and Create Data frame.

Access Azure Preview Portal (*Ignore this section if you're provided with a shared cluster*)

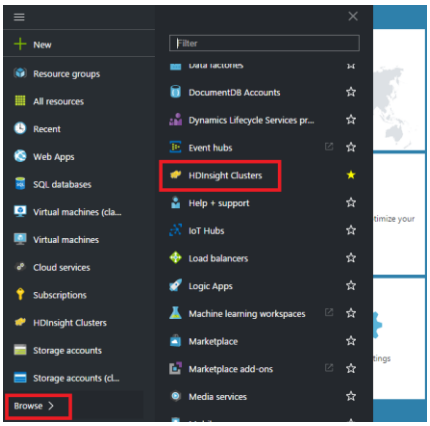
1. Sign in to the Azure preview portal.

If Spark Cluster pinned to StartBoard

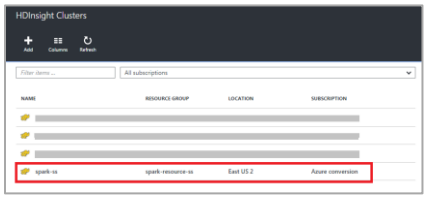
2. Click tile for your Spark Cluster.

If not pinned to StartBoard

2. Click **Browse,** select **HDInsight Clusters**.

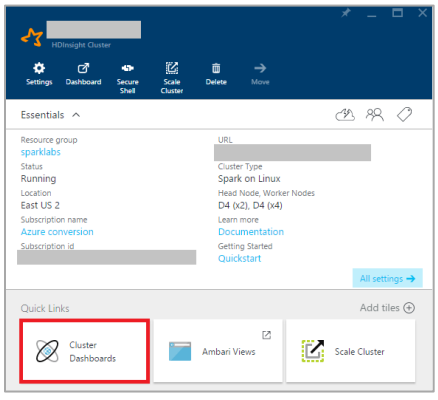


3. Select your Spark Cluster.

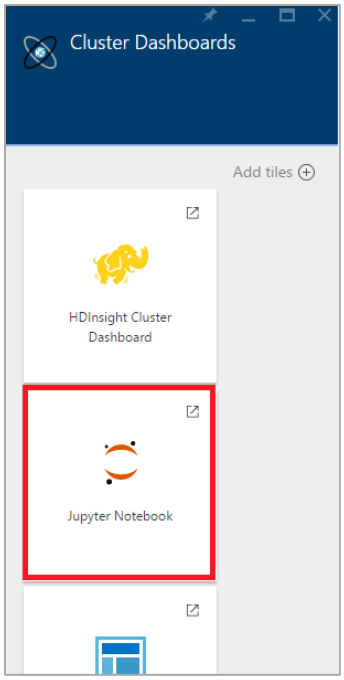


Launch Jupyter Notebook (*Ignore this section if you're provided with a shared cluster)*

1. Click on **Cluster Dashboards** tile present under **Quick Links** of Cluster Blade.



2. Locate the **Jupyter Notebook** tile on Cluster Dashboards tile array and click on it.



Launch Jupyter Notebook on shared cluster (*Use the scredentials in this section if you're provided with a shared cluster)*

Jupyter Tree URL: https://nthdilabsspark.azurehdinsight.net/jupyter/tree

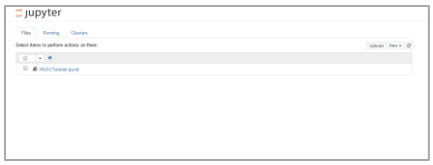
Username: admin

Password: HDItut@123

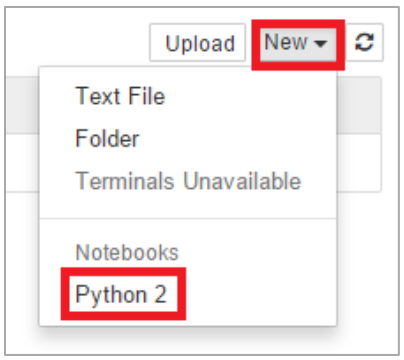
Create a new Jupyter Notebook

When prompted, enter the admin credentials for the Spark cluster.

Jupyter Notebook will open.



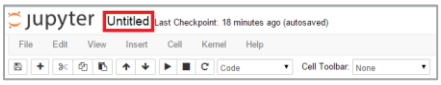
1. Click **New** dropdown button at top right side of Jupyter Notebook screen.
2. Click **Python 2** under Notebooks, from the dropdown.



Assign friendly name to notebook

A new notebook is then created and opened with the name **Untitled.pynb.**

1. Click the name of the notebook at the top to rename it.



2. Enter new name as **SparkSQL-Lab01** and press button **OK**.

Create Spark and SQL context

1. The statement "`from` *namespace* `import Class_Name`" imports classes from defined namespaces, required for execution of jobs
2. `SparkContext` ('yarn-client')
3. Above line initializes the spark context and launches Spark application on YARN in client mode.
4. `SQLContext(sparkContext)`
5. Above line of code initializes the sql context. Constructor `SQLContext` accepts `SparkContext object as a parameter.`
6. The **atexit** module defines functions to register and unregister cleanup functions. Functions thus registered are automatically executed upon normal interpreter termination.

Import the required modules and create the Spark and SQL contexts.

1. Paste the following snippet in an empty cell.
2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside the cell.

```
from pyspark import SparkContext
from pyspark.sql import SQLContext
import atexit
from pyspark.sql.types import *
```

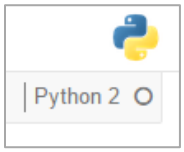
```
sqlc = SQLContext(sc)
atexit.register(lambda: sc.stop())
```

```
from pyspark import SparkContext
from pyspark.sql import SQLContext
import atexit
from pyspark.sql.types import *

sc = SparkContext('yarn-client')
sqlc = SQLContext(sc)
atexit.register(lambda: sc.stop())
```

Create data frame from data stored in azure blob storage

1. Every time you run a job in Jupyter, your web browser window title will show a (Busy) status alongside the notebook title.
2. You will also see a solid circle next to the Python 2 text in the top-right corner.

3. After the job completes, this will change to a hollow circle



1. customerData = sc.textFile("File_Path")

   sc.textFile reads the text file line by line at the specified location.

2. customerData.map(lambda c:c.split(",")) splits each line in customerData variable, using "," as delimiter and filter(lambda s:s[0]!="CustomerId") helps to filter out header line
3. StructField("Column_Name", DataType Boolean) defines column and it's data type
4. StructType(List_of_StructFields) creates a schema for data frame using collection of StructField objects.
5. customerDataFinal.toDF(customerDataSchema) converts customerDataFinal rdd object to dataframe using customerDataSchema schema object

1. Paste the following snippet in below empty cell
2. Replace **<Container_Name>** in code with name of blob container containing dataset file to be accessed.
3. Replace **<Your_Storage_Account_Name>** in code with Azure storage account name
4. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.

```
customerData =
sc.textFile("wasb://nthdilabsspark@nthdilabs.blob.core.windows.net/<your_name>/SparkLabDa
tasets/Lab01/CustomerData.csv")
customerDataFinal = customerData.map(lambda c:c.split(",")).filter(lambda
s:s[0]!="CustomerId")
```

```
customerDataFields = [StructField("Id", StringType(), True),
                      StructField("Name", StringType(), True),
                      StructField("City", StringType(), True),
                      StructField("State", StringType(), True),
                      StructField("Email", StringType(), True),
                      StructField("Phone", StringType(), True)]


customerDataSchema = StructType(customerDataFields)

customerData_DataFrame = customerDataFinal.toDF(customerDataSchema)

customerData_DataFrame.head(10)
```

5. Output of above code execution will be as shown below:

```
[Row(Id=u'JRK07IRCIJ', Name=u'Anne Fernandez', City=u'Charlotte', State=u'NC',
Email=u'annef@yahoo.com', Phone=u'212-776-4435'),
 Row(Id=u'ZJX5W46O91', Name=u'Adam Brown', City=u'Fremont', State=u'CA',
Email=u'adamb@googlemail.com', Phone=u'214-778-6654'),
 Row(Id=u'1AYIINRJT0', Name=u'Rob Wilson', City=u'Michigan City', State=u'MI',
Email=u'rob.wilson@gmail.com', Phone=u'206-877-8996'),
.
.
.]
```

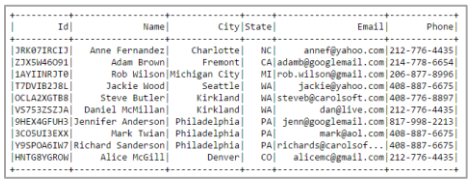
Output similar to this signifies successful creation of data frame for customer data.

DataFrame operations

Execute following operations on DataFrame created earlier in this lab and observe the output. Use Empty cells in the notebook to execute these operations.

1. customerData_DataFrame.show()

Output:



Id	Name	City	State	Email	Phone
JRK07IRCIJ	Anne Fernandez	Charlotte	NC	annef@yahoo.com	212-776-4435
ZJX5W46O91	Adam Brown	Fremont	CA	adamb@googlemail.com	214-778-6654
1AYIINRJT0	Rob Wilson	Michigan City	MI	rob.wilson@gmail.com	206-877-8996
T7DVIB2J8L	Jackie Wood	Seattle	WA	jackie@yahoo.com	408-887-6675
OCLA2XGTB8	Steve Butler	Kirkland	WA	steveb@carolsoft.com	408-776-8897
V5753ZSZJA	Daniel McMillan	Kirkland	WA	dan@live.com	212-776-4435
9HEX4GFUH3	Jennifer Anderson	Philadelphia	PA	jenn@googlemail.com	817-998-2213
3CO5UI3EXX	Mark Twian	Philadelphia	PA	mark@aol.com	408-887-6675
Y9SPOA6IW7	Richard Sanderson	Philadelphia	PA	richards@carolsof...	408-887-6675
HNTG8VGROW	Alice McGill	Denver	CO	alicemc@gmail.com	212-776-4435

2. customerData_DataFrame.select("Name").show()

Output:

```
+------------------+
|              Name|
+------------------+
|   Anne Fernandez|
|        Adam Brown|
|        Rob Wilson|
|       Jackie Wood|
|      Steve Butler|
|  Daniel McMillan|
|Jennifer Anderson|
|        Mark Twian|
|Richard Sanderson|
|      Alice McGill|
+------------------+
```

3. customerData_DataFrame.filter(customerData_DataFrame.Name == 'Alice McGill').show()

Output:

```
+---------+-----------+------+-----+----------------+-----------+
|       Id|       Name|  City|State|           Email|      Phone|
+---------+-----------+------+-----+----------------+-----------+
|HNTG8YGROW|Alice McGill|Denver|   CO|alicemc@gmail.com|212-776-4435|
+---------+-----------+------+-----+----------------+-----------+
```

4. customerData_DataFrame.groupBy("State").count().show()

Output:

```
+-----+-----+
|State|count|
+-----+-----+
|   NC|    1|
|   CA|    1|
|   PA|    3|
|   CO|    1|
|   WA|    3|
|   MI|    1|
+-----+-----+
```

Running SQL Queries

1. To register the DataFrame as SQL table copy below code in empty cell and execute it

```
customerData_DataFrame.registerTempTable("CustomerTable")
```

2. Execute below SQL queries in empty cells and observe the output

```
%%sql
Select * from CustomerTable
```

Output:

```
+---------+---------------+-----------+-----+-----------------+-----------+
|       Id|           Name|       City|State|            Email|      Phone|
+---------+---------------+-----------+-----+-----------------+-----------+
|7RK07IRCIJ|  Anne Fernandez|  Charlotte|   NC|     annef@yahoo.com|212-776-4435|
|I3XSW46O91|     Adam Brown|    Fremont|   CA|adamb@googlemail.com|214-778-6654|
|1AYIINRJT0|     Rob Wilson|Michigan City|   MI|rob.wilson@gmail.com|206-877-8996|
|T7DVIBZJ8L|    Jackie Wood|    Seattle|   WA|    jackie@yahoo.com|408-887-6675|
|OCLA2XGTB8|    Steve Butler|    Kirkland|   WA|steveb@carolsoft.com|408-776-8897|
|V5753ZSZJA| Daniel McMillan|    Kirkland|   WA|        dan@live.com|212-776-4435|
|9HEX4GF1H3|Jennifer Anderson|Philadelphia|   PA| jenn@googlemail.com|817-998-2213|
|3CO5UI3EXX|      Mark Twian|Philadelphia|   PA|       mark@aol.com|408-887-6675|
|Y9SPOA6IW7|Richard Sanderson|Philadelphia|   PA|richards@carolsof...|408-887-6675|
|HNTG8YGROW|     Alice McGill|     Denver|   CO|   alicemc@gmail.com|212-776-4435|
+---------+---------------+-----------+-----+-----------------+-----------+
```

```
%%sql
Select Name from CustomerTable order by Name
```

Output:

```
+-----------------+
|             Name|
+-----------------+
|       Adam Brown|
|     Alice McGill|
|   Anne Fernandez|
|  Daniel McMillan|
|      Jackie Wood|
|Jennifer Anderson|
|       Mark Twian|
|Richard Sanderson|
|       Rob Wilson|
|     Steve Butler|
+-----------------+
```

```
%%sql
Select Name from CustomerTable where State='WA'
```

Output:

```
+---------------+
|           Name|
+---------------+
|    Jackie Wood|
|   Steve Butler|
|Daniel McMillan|
+---------------+
```

```
%%sql
Select State,count(Name) as CNT from CustomerTable group by State
```

Output:

```
+-----+---+
|State|CNT|
+-----+---+
|   NC|  1|
|   CA|  1|
|   PA|  3|
|   CO|  1|
|   WA|  3|
|   MI|  1|
+-----+---+
```

Section 4: Understand joins, functions and user defined functions

Create New Jupyter Notebook

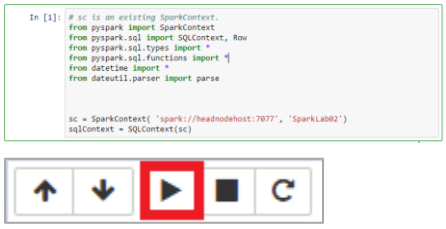
Create Spark and SQL context

Import the required modules and create the Spark and SQL contexts.

1. Paste the following snippet in an empty cell
2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.

```
from pyspark import SparkContext
```

```
from pyspark.sql import SQLContext
import atexit
from pyspark.sql.types import *
from pyspark.sql.functions import *

sqlc = SQLContext(sc)
atexit.register(lambda: sc.stop())
```



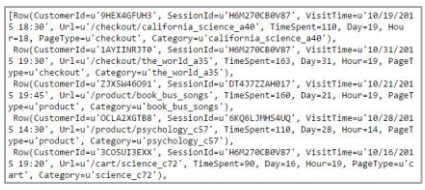
Create Data Frame for Datasets

Create Data Frame for customer data set

1. Paste the following snippet in an empty cell.
2. **<azure_storage_account>** placeholder with name of storage account used to store datasets and replace **<blob_container>** with name of container containing datasets.
3. Press **SHIFT + ENTER**. Or Press **Play** button from tool bar to execute the code inside cell.

```
customerData =
sc.textFile("wasb://nthdilabsspark@nthdilabs.blob.core.windows.net/<yourname>/SparkLabDat
asets/Lab01/CustomerData.csv")
customerDataFinal = customerData.map(lambda c:c.split(",")).filter(lambda
s:s[0]!="CustomerId")

customerDataFields = [StructField("Id", StringType(), True),
                      StructField("Name", StringType(), True),
                      StructField("City", StringType(), True),
                      StructField("State", StringType(), True),
                      StructField("Email", StringType(), True),
                      StructField("Phone", StringType(), True)]

customerDataSchema = StructType(customerDataFields)

customerDF = customerDataFinal.toDF(customerDataSchema)

customerDF.head(10)
```

4. Output similar to belwo figure denotes successful data frame creation



Create Data Frame for transaction data set

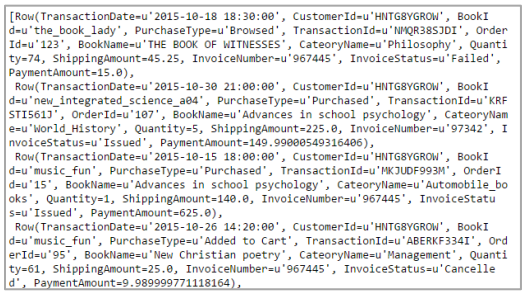
1. Paste the following snippet in an empty cell.

2. **<azure_storage_account>** placeholder with name of storage account used to store datasets and replace **<blob_container>** with name of container containing datasets.

3. Press **SHIFT + ENTER**. Or Press **Play** button from tool bar to execute the code inside cell.

```
transactionData =
sc.textFile("wasb://nthdilabsspark@nthdilabs.blob.core.windows.net/<yourname>/SparkLabDat
asets/Lab01/TransactionData.csv")
transactionDataRaw = transactionData.map(lambda c:c.split(",")).filter(lambda
s:s[0]!="TransactionDate")

transactionHeaders = transactionData.first()
transactionFields = [StructField("TransactionDate",StringType(),True),
                     StructField("CustomerId",StringType(),True),
                     StructField("BookId",StringType(),True),
                     StructField("PurchaseType",StringType(),True),
                     StructField("TransactionId",StringType(),True),
                     StructField("OrderId",StringType(),True),
                     StructField("BookName",StringType(),True),
                     StructField("CateoryName",StringType(),True),
                     StructField("Quantity",IntegerType(),True),
                     StructField("ShippingAmount",FloatType(),True),
                     StructField("InvoiceNumber",StringType(),True),
                     StructField("InvoiceStatus",StringType(),True),
                     StructField("PaymentAmount",FloatType(),True)
                     ]

transactionSchema = StructType(transactionFields)
transactionDataFinal = transactionDataRaw.map(lambda p:
(p[0],p[1],p[2],p[3],p[4],p[5],p[6],p[7],int(p[8]),float(p[9]),p[10],p[11],float(p[12])))
transactionsDF = transactionDataFinal.toDF(transactionSchema)
transactionsDF.head(10)
```

4. Output similar to below figure denotes successful data frame creation



Create Data Frame for session data set

1. Paste the following snippet in an empty cell.

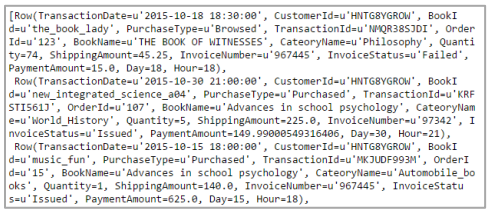
2. Replace **<azure_storage_account>** placeholder with name of storage account used to store datasets and replace **<blob_container>** with name of container containing datasets.

3. Press **SHIFT + ENTER**. Or Press **Play** button from tool bar to execute the code inside cell.

```
sessionData =
sc.textFile("wasb://nthdilabsspark@nthdilabs.blob.core.windows.net/<yourname>/SparkLabDat
asets/Lab01/SessionData.csv")
sessionDataRaw = sessionData.map(lambda c:c.split(","))
```

```
sessionFields = [StructField("CustomerId",StringType(),True),
                 StructField("SessionId",StringType(),True),
                 StructField("VisitTime",StringType(),True),
                 StructField("Url",StringType(),True),
                 StructField("TimeSpent",IntegerType(),True)
                ]

sessionSchema = StructType(sessionFields)
sessionDataFinal = sessionDataRaw.map(lambda p: (p[0],p[1],p[2],p[3],int(p[4])))
sessionsDF = sessionDataFinal.toDF(sessionSchema)
sessionsDF.head(10)
```

4. Output similar to below figure denotes successful data frame creation



Modify transaction data frame to add calculated columns

1. Transaction Date is in **"yyyy:mm:dd hh:mm:ss"** string format. For data analysis we will split above string to get Day and Hour and put them in separate columns "**Day**" and "**Hour**" in integer format respectively.
2. To achieve this we have to create user defined functions using the format as follows:

```
def py_function_name(argument_list)
#code
return return_parameter
udf_name = udf(py_function_name, data_type)
```

3. Before using **udf** function, **udf** class should be imported to notebook

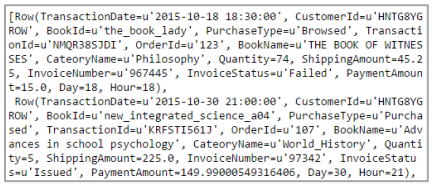
```
E.g.: from pyspark.sql.functions import udf
```

4. Paste the following snippet in an empty cell.
5. Press **SHIFT + ENTER**. Or Press Play button from tool bar to execute the code inside cell.

```
from pyspark.sql.functions import udf
# function to extract hour from string date time
def functionGetHour(Date):

    split1 = Date.split(" ")[1]
    Hour = split1.split(":")[0]
    return int(Hour)

getHour = udf(functionGetHour, IntegerType())

# function to extract day from string date time

def functionGetDay(Date):

    split1 = Date.split("-")[2]
    Day = split1.split(" ")[0]
    return int(Day)
```

```
getDay = udf(functionGetDay, IntegerType())

transactionsDF01 = transactionsDF.select(
                                    "*",
                                    getDay(transactionsDF.TransactionDate).alias('Day'),
                                    getHour(transactionsDF.TransactionDate).alias('Hour'))

transactionsDF01.head(10)
```

6. Output similar to below figure denotes successful data frame creation



Modify session data frame to add calculated columns

1. Similar to transaction data frame, we will create two columns in session data frame, for Day and Hour by splitting date time string in VisitTime column.
2. We will also write two separate functions to extract page type and category of book from Url of data and place it in **PageType** and **Category** column respectively.
3. Paste the following snippet in an empty cell.
4. Press **SHIFT + ENTER**. Or Press Play button from tool bar to execute the code inside cell.

```
# function to extract Day from date time string
def functionGetDayS(Date):

    Day = Date.split("/")[1]

    return int(Day)

getDayS = udf(functionGetDayS, IntegerType())

# function to extract page type from url
def functionGetPageType(URL):

    PageType = URL.split("/")[1]
    return PageType

getPageType = udf(functionGetPageType, StringType())

# function to extract category from url

def functionGetCategory(URL):

    Category = URL.split("/")[2]
    return Category

getCategory = udf(functionGetCategory, StringType())


sessionsDF01 = sessionsDF.select(
                                    "*",
                                    getDayS(sessionsDF.VisitTime).alias('Day'),
                                    getHour(sessionsDF.VisitTime).alias('Hour'),
```

```
                                    getPageType(sessionsDF.Url).alias('PageType'),
                                    getCategory(sessionsDF.Url).alias('Category')
                                    )

sessionsDF01.head(10)
```

5. Output similar to below figure denotes successful data frame creation

Join transaction and session data frames with customer data frame

1. Paste the following snippet in an empty cell to join transaction and session data frame with customer data frame.
2. Press **SHIFT + ENTER**. Or Press **Play** button from tool bar to execute the code inside cell.

```
#perform joins on session and customer data frame
customerSessionDF = sessionsDF01.join(customerDF,sessionsDF01.CustomerId ==
customerDF.Id)

#perform joins on transaction and customer data frame
customerTransactionDF = transactionsDF01.join(customerDF,transactionsDF01.CustomerId ==
customerDF.Id)
```

Perform operations on data frames to analyze the data

Call functions on data frames to analyze the data

- **groupBy(*cols)** : Groups the DataFrame using specified columns, inorder to run aggregation on them.
- **count():** Returns the number of rows in DataFrame.
- **collect():** Returns all records as list of row.
- **orderBy(*cols):** Returns a new DataFrame sorted by the specified columns.
- **desc():** Returns a sort expression based on the descending order of the given column name.
- **avg(*args):** Computes average values for each numeric column for each group.
- **sum(*args):** Computes sum for each numeric column for each group.

1. Paste the following queries one by one in the empty cells and execute them to observe their output.
2. Press **SHIFT + ENTER**. Or Press **Play** button from tool bar to execute the code inside cell.

3. Get number of sessions created by customers

```
Query:
customerSessionDF.groupBy("Name").count().collect()
```

Output:

```
[Row(Name=u'Jennifer Anderson', count=27984),
 Row(Name=u'Rob Wilson', count=28036),
 Row(Name=u'Alice McGill', count=27897),
 Row(Name=u'Mark Twian', count=27913),
 Row(Name=u'Steve Butler', count=28052),
 Row(Name=u'Anne Fernandez', count=28152),
 Row(Name=u'Jackie Wood', count=27987),
 Row(Name=u'Daniel McMillan', count=27901),
 Row(Name=u'Richard Sanderson', count=27866),
 Row(Name=u'Adam Brown', count=28212)]
```

4. Get names of first three customers who created maximum sessions.

```
Query:
customerSessionDF.groupBy("Name").count().orderBy("count",ascending=False).head(3)
```

Output:

```
[Row(Name=u'Adam Brown', count=28212),
 Row(Name=u'Anne Fernandez', count=28152),
 Row(Name=u'Steve Butler', count=28052)]
```

5. Get average time spent by customer on each session.

```
Query:
customerSessionDF.groupBy().avg("TimeSpent").collect()
```

Output:

```
[Row(AVG(TimeSpent)=145.43445714285716)]
```

6. Get the name of customer who spent maximum time and total time spent.

Query:

```
customerSessionDF.groupBy("Name").sum("TimeSpent").orderBy("sum(TimeSpent)",ascendi
ng=False).head(1)
```

Output:

```
[Row(Name=u'Adam Brown', SUM(TimeSpent)=4103352)]
```

7. Get day wise session frequency.

Query:

```
customerSessionDF.groupBy("Day").count().collect()
```

Output:

```
[Row(Day=31, count=23474),
 Row(Day=12, count=23426),
 Row(Day=15, count=23363),
 Row(Day=16, count=23484),
 Row(Day=19, count=23454),
 Row(Day=20, count=23426),
 Row(Day=21, count=46306),
 Row(Day=23, count=23363),
 Row(Day=25, count=23253),
 Row(Day=28, count=23054),
 Row(Day=30, count=23397)]
```

Output denotes session frequency is same for all days **except 21st due to offers provided on bookstore site**.

8. Get three most browsed urls.

Query:

```
customerSessionDF.groupBy("Url").count().orderBy("count", ascending=False).head(3)
```

Output:

```
[Row(Url=u'/product/psychology_c57', count=28213),
 Row(Url=u'/product/spotlight_books', count=28139),
 Row(Url=u'/cart/driving_jarvis_ham', count=28078)]
```

Perform analysis on purchased book data

Where(condition): Filters rows based on provided condition.

1. Transaction data contains column PurchaseType, it has three values Browsed, Purchased and Added to Cart.
2. To perform operations on purchased books, filter transaction data to create new DataFrame having only purchased data.
3. Paste the following code snippet in empty cell to create DataFrame for purchased data
4. Press **SHIFT + ENTER**. Or Press Play button from tool bar to execute the code inside cell.

```
purchasedDF = customerTransactionDF.where(transactionsDF01.PurchaseType =='Purchased')
```

withColumn(colName, colExpr): Returns a new data frame by adding column.

1. Perform following operations one by one in empty cells and observe the output
2. Find out five most purchased book categories

```
Query:
purchasedDF.groupBy("CateoryName").sum("Quantity").orderBy("sum(Quantity)",ascendin
g=False).show(5)
```

Output:

```
CateoryName   SUM(Quantity)
Drive_books   211029
Adventure     112470
World_History 112263
Art           112105
Non_Fiction   111731
```

3. Find out most purchased books

Query:

```
purchasedDF.groupBy("BookName").sum("Quantity").orderBy("sum(Quantity)",
ascending=False).show(5)
```

Output:

```
BookName             SUM(Quantity)
The voyages of Ca... 232414
Advances in schoo... 231410
Science in Dispute   231408
History of politi... 231255
THE BOOK OF WITNE... 230872
```

4. Add a new calculated column to purchasedDF DataFrame having name "AmountPaid" and containing value

```
AmountPaid = (Quantity * PaymentAmount) + ShippingAmount
```

And find out top five customers who paid most.

Query:

```
purchasedDF01 = purchasedDF.withColumn("AmountPaid",(purchasedDF.Quantity *
purchasedDF.PaymentAmount) + purchasedDF.ShippingAmount)
purchasedDF01.groupBy("Name").sum("AmountPaid").orderBy("sum(AmountPaid)",ascending
=False).select("Name").show(5)
```

Output:

```
Name
Anne Fernandez
Steve Butler
Jennifer Anderson
Alice McGill
Rob Wilson
```

Get the amount collected on 18th day

Query:

```
purchasedDF01.where(purchasedDF01.Day == 18).groupBy().sum("AmountPaid").collect()
```

Output:

```
[Row(SUM(AmountPaid)=49105849.035095215)]
```

Disclaimer: Once you have completed the lab, to reduce costs associated with your Azure subscription, you may want to delete your clusters.

For TechReady, please leave the cluster as they are, you may choose to delete your notebooks.

Terms of use

By using this hands-on lab, you agree to the following terms:

The technology/functionality described in this hands-on lab is provided by Microsoft Corporation in a "sandbox" testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the hands-on lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this hands-on lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCTIONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

FEEDBACK. If you give feedback about the technology features, functionality, and/or concepts described in this hands-on lab to Microsoft, you give to Microsoft, without charge, the right to use, share, and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies, and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED, OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.