Microsoft

Apache Spark on Microsoft Azure HDInsight: Using Spark MLLib in 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	7/17/2017	Ready 2017	Nishant Thacker

Contents

Introduction

This class specifically focuses on Spark ML component of Spark and highlights its value proposition in the Apache Spark Big Data processing framework.

Spark ML is Apache Spark's scalable machine learning library.

The main highlights of Spark ML are as follows (source: https://spark.apache.org/mllib/):

# 1 Ease of Use.

Spark ML fits into Spark's APIs and one can use any Hadoop data source (e.g. HDFS, HBase, or local files), making it easy to plug into Hadoop workflows. Usable in Java, Scala, Python, and SparkR.

#2 Performance.

Spark ML runs high-quality algorithms about 100x faster than MapReduce.

#3 Easy to Deploy.

Spark ML, along with Spark runs on existing Hadoop clusters and data.

Business Use Case

Product Recommendations and Personalized Marketing for a retail website. For ex: ‘people like you also like this’.

Takeaways

The students will learn how online retailers provide product recommendations and market personalized items to them, while they are shopping online.

They will learn the machine learning algorithms, understand the data sets required to perform these computations and the overall process flow that brings it all together from a site visitor's perspective.

The ‘People Like You Also Like This’ use case: The algorithm to enable this functionality is implemented in SparkML and explained here:

Collaborative Filtering

http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

This algorithm looks deep into site visitor's profile, previous browsing and shopping history. Additionally, it looks into the sales history of current items in shopping basket, general sales history of other items bought together with the current item, current item's product characteristics and properties and develop correlation's with other products in their catalog.

TechReady24 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.

TechReady24 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 Ready you're provided a cluster with the following credentials:*

<u>Spark Cluster:</u>

Cluster Name for Spark Cluster: nthdilabsspark

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

Username: admin

Password: HDItut@123

<u>Azure Credentials, if needed:</u>

Username: analyticsdemo@outlook.com

Password: HDItut@123

Class

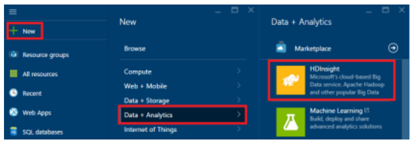
Section 1: Provision an HDInsight Spark cluster

Access Azure Preview Portal

1. Sign in to the Azure preview portal.

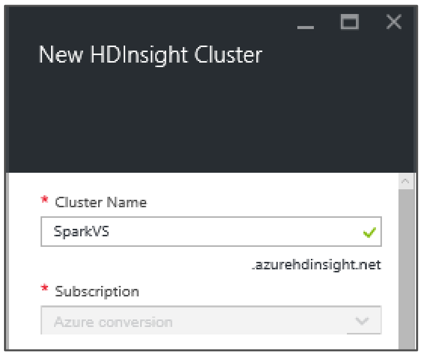
Create HDInsight Spark cluster

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



Provide Cluster Details

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



New HDInsight Cluster

* Cluster Name

SparkVS

.azurehdinsight.net

* Subscription

Azure conversion

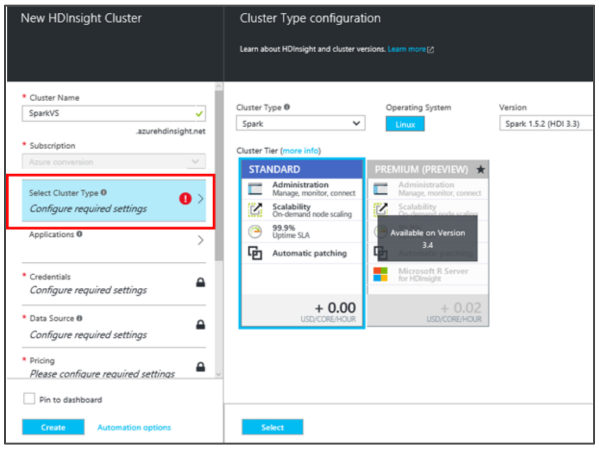
A green check mark appears beside the cluster name if it is available.

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.

1.

Configure Cluster Type

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



New HDInsight Cluster

* Cluster Name: SparkVS .azurehdinsight.net

* Subscription: Azure conversion

Select Cluster Type — Configure required settings

Applications

* Credentials — Configure required settings

* Data Source — Configure required settings

* Pricing — Please configure required settings

Pin to dashboard

Create | Automation options

Cluster Type configuration

Learn about HDInsight and cluster versions. Learn more

Cluster Type: Spark | Operating System: Linux | Version: Spark 1.5.2 (HDI 3.3)

Cluster Tier (more info)

STANDARD: Administration (Manage, monitor, connect); Scalability (On-demand node scaling); 99.9% Uptime SLA; Automatic patching; + 0.00 USD/CORE/HOUR

PREMIUM (PREVIEW): Administration (Manage, monitor, connect); Scalability (On-demand node scaling); Automatic patching; Microsoft R Server for HDInsight; + 0.02 USD/CORE/HOUR

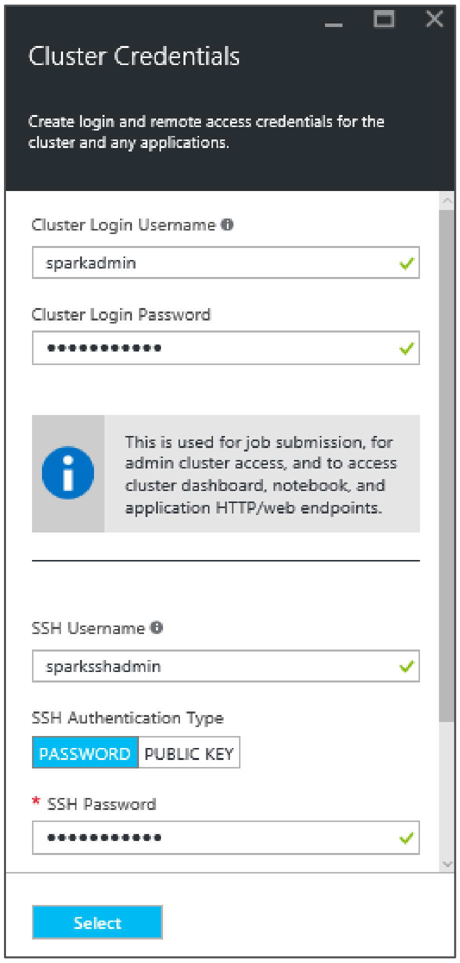
Available on Version 3.4

Select

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.**



Cluster Credentials

Create login and remote access credentials for the cluster and any applications.

Cluster Login Username: sparkadmin

Cluster Login Password: ●●●●●●●●●●●●

This is used for job submission, for admin cluster access, and to access cluster dashboard, notebook, and application HTTP/web endpoints.

SSH Username: sparksshadmin

SSH Authentication Type: [✓ PASSWORD] PUBLIC KEY

* SSH Password: ●●●●●●●●●●●●

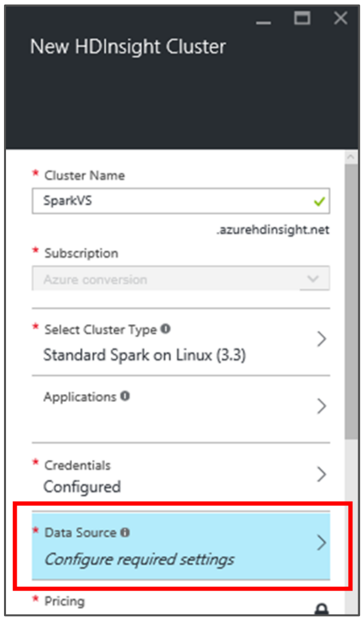
2. Enter **Cluster Login Username.**
3. Enter **Cluster Login Password**.
4. Enter **SSH Username**.
5. Select **PASSWORD** as **SSH Authentication Type**.
6. Enter **SSH Password** and confirm it.
7. Click **Select** button to save the credentials.

**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


New HDInsight Cluster

* Cluster Name: SparkVS .azurehdinsight.net

* Subscription: Azure conversion

* Select Cluster Type: Standard Spark on Linux (3.3)

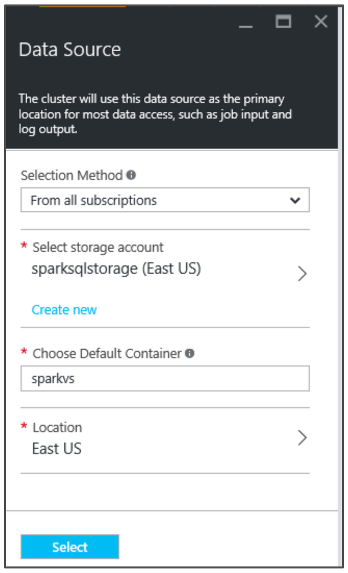
Applications

* Credentials: Configured

* Data Source: *Configure required settings*

* Pricing

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


Data Source

The cluster will use this data source as the primary location for most data access, such as job input and log output.

Selection Method: From all subscriptions

* Select storage account: sparksqlstorage (East US)

Create new

* Choose Default Container: sparkvs

* Location: East US

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

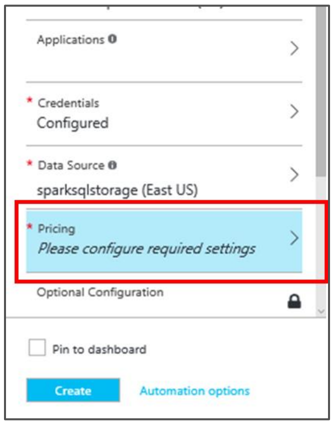
   Or
   Click on link **Select Existing** to select from existing accounts.
   For purspose of this exercise, we will create a new storage account.
4. Enter name for default container to be designated for cluster in **Choose Default Container** field**.**
5. If existing storage account is selected then 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.



Applications

Credentials
Configured

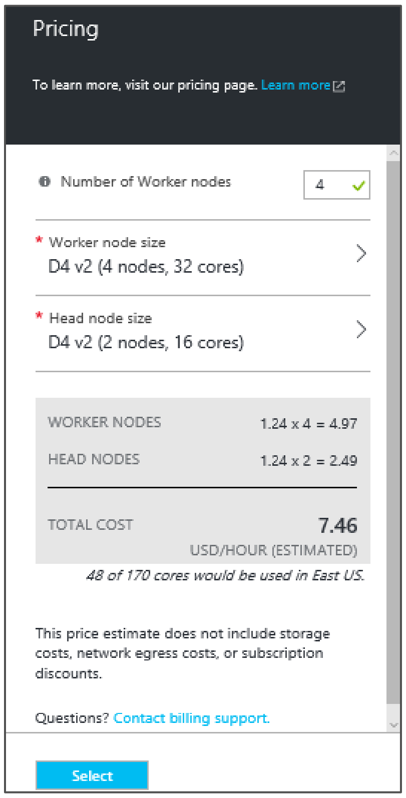
Data Source
sparksqlstorage (East US)

Pricing
Please configure required settings

Optional Configuration

Pin to dashboard

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.



Pricing

To learn more, visit our pricing page. Learn more

Number of Worker nodes: 4

Worker node size
D4 v2 (4 nodes, 32 cores)

Head node size
D4 v2 (2 nodes, 16 cores)

WORKER NODES	1.24 x 4 = 4.97
HEAD NODES	1.24 x 2 = 2.49
TOTAL COST	7.46 USD/HOUR (ESTIMATED)

48 of 170 cores would be used in East US.

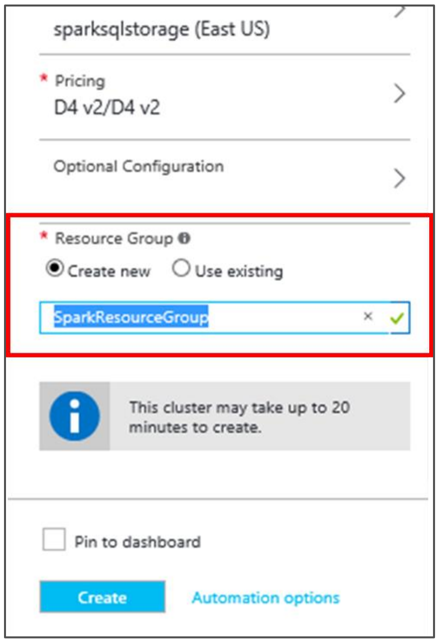
This price estimate does not include storage costs, network egress costs, or subscription discounts.

Questions? Contact billing support.

3. Leave all other values as default.
2. 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:


sparksqlstorage (East US)

* Pricing
D4 v2/D4 v2

Optional Configuration

* Resource Group
[✓ Create new] Use existing
SparkResourceGroup

This cluster may take up to 20 minutes to create.

☐ Pin to dashboard

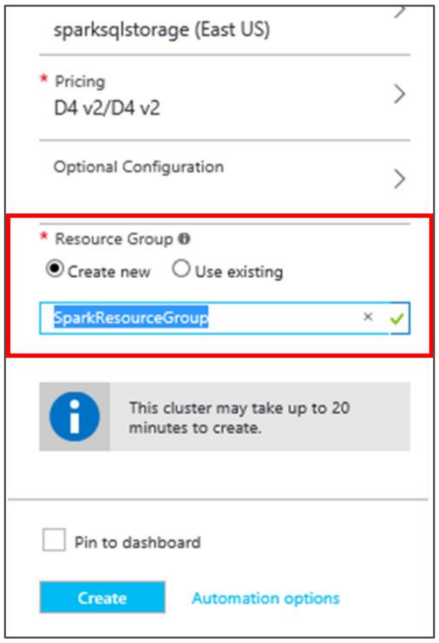
a) Click link Create New to provide new Resource Group.

Or

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

3. 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.


sparksqlstorage (East US)

* Pricing
D4 v2/D4 v2

Optional Configuration

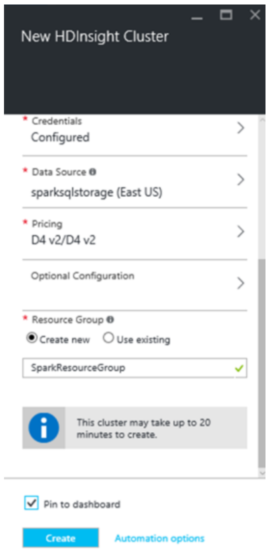
* Resource Group
[✓ Create new] Use existing
SparkResourceGroup

This cluster may take up to 20 minutes to create.

☐ Pin to dashboard

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.


New HDInsight Cluster

* Credentials
Configured

* Data Source
sparksqlstorage (East US)

* Pricing
D4 v2/D4 v2

Optional Configuration

* Resource Group
[✓ Create new] Use existing

SparkResourceGroup

This cluster may take up to 20 minutes to create.

[✓] Pin to dashboard

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.

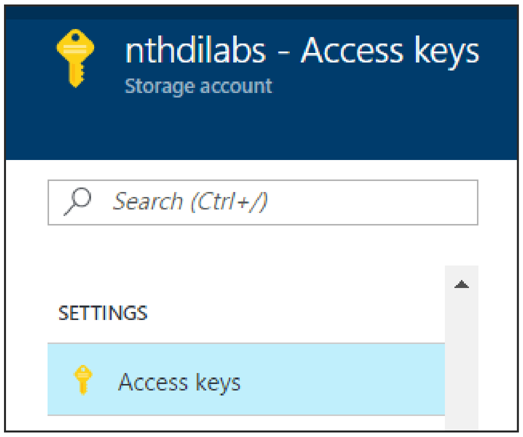
Section 2: Load datasets files to storage account.

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

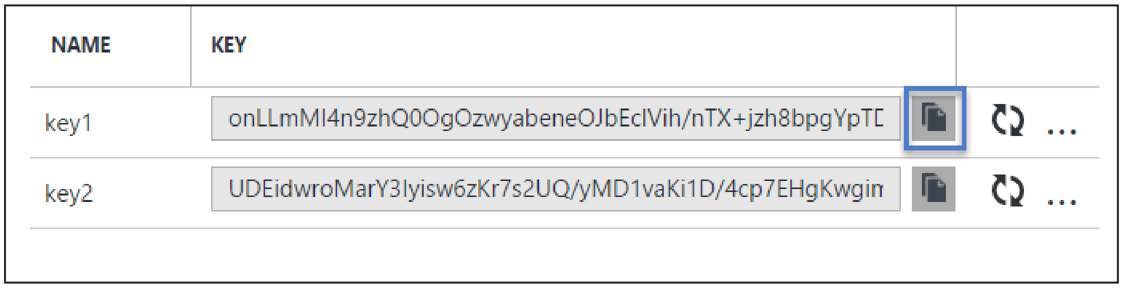
Note: Ignite Step 1 and 2 if you are provided a shared cluster (for TechReady)

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.



NAME	KEY
key1	onLLmMl4n9zhQ0OgOzwyabeneOJbEclVih/nTX+jzh8bpgYpTD
key2	UDEidwroMarY3lyisw6zKr7s2UQ/yMD1vaKi1D/4cp7EHgKwgin

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/<yourclustername>/<yourname>/Spark
LabDatasets/
/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.

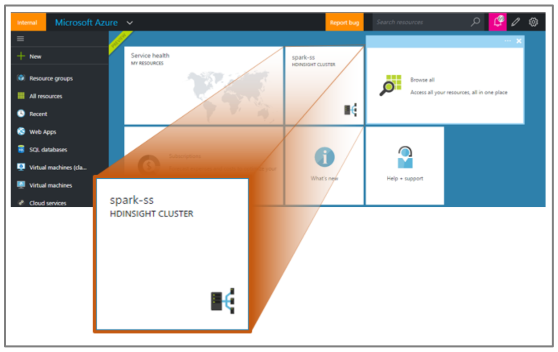
Section 3: Creating a new Jupyter Notebook

Access Azure Portal

1. Sign in to the Azure Portal.

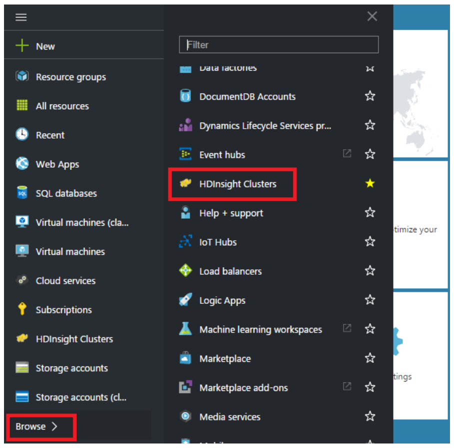
If Spark Cluster is pinned to the "StartBoard":

2. Click the tile for your Spark Cluster.

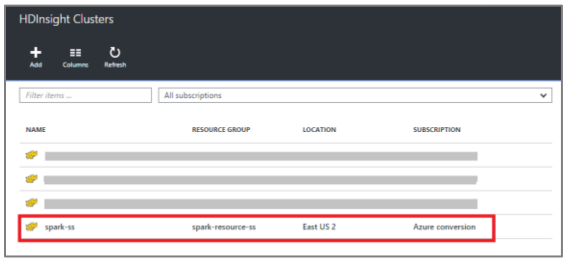


If Spark Cluster is not pinned to the “StartBoard”:

1. Click Browse, select HDInsight Clusters.



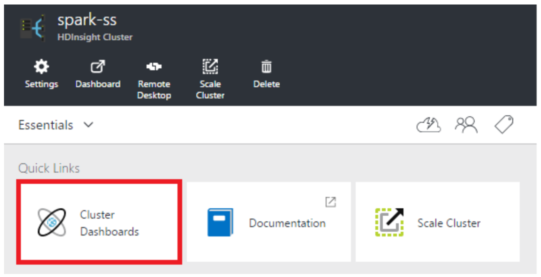
2. Select your Spark Cluster.



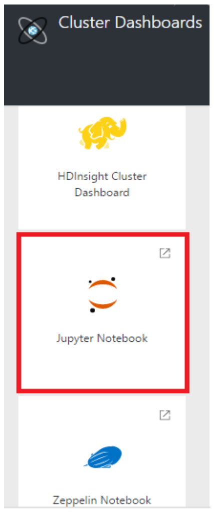
NAME	RESOURCE GROUP	LOCATION	SUBSCRIPTION
spark-ss	spark-resource-ss	East US 2	Azure conversion

Launch Jupyter Notebook

1. Click on Cluster Dashboards tile displayed under the Quick Links of Cluster Blade.

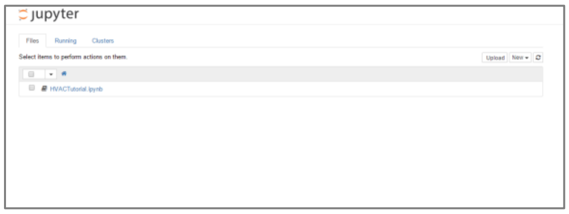


2. Locate **Jupyter Notebook** tile on Cluster Dashboards tile and click on it.



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

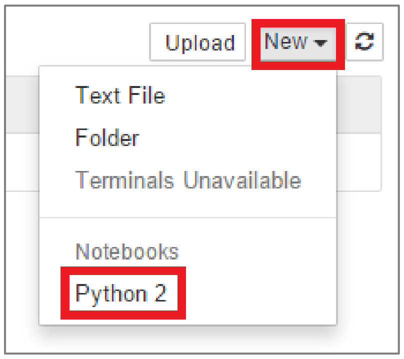
This will open the Jupyter dashboard.



Select items to perform actions on them.

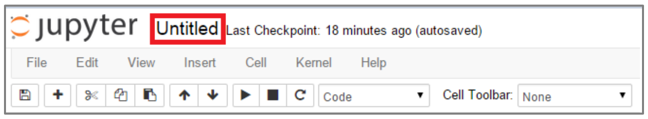
Create a new notebook

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



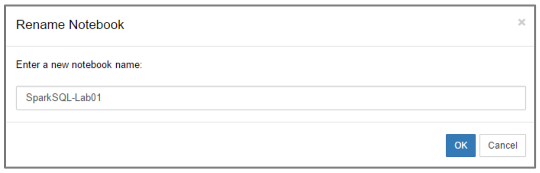
Assign a friendly name to the newly created notebook

1. A new notebook is created and has the default name: **Untitled.pynb.**
2. Click the notebook's name at the top to rename it.



Last Checkpoint: 18 minutes ago (autosaved)

3. Enter new notebook name as SparkSQL-Lab02 and press OK


Rename Notebook

Enter a new notebook name:

SparkSQL-Lab01

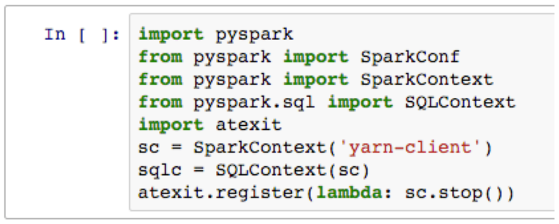
Create the Spark context

The atexit module defines functions to register and unregister cleanup functions. Functions thus registered are automatically executed upon normal interpreter termination.

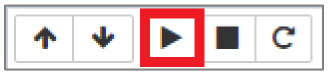
Import the required modules and create the Spark contexts.

1. Paste the following snippet in an empty cell.

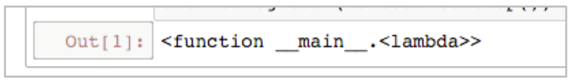
```
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SQLContext
import atexit

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


```
In [ ]: import pyspark
        from pyspark import SparkConf
        from pyspark import SparkContext
        from pyspark.sql import SQLContext
        import atexit
        sc = SparkContext('yarn-client')
        sqlc = SQLContext(sc)
        atexit.register(lambda: sc.stop())
```

Here you are creating a SparkContext object, which is the main entry point for Spark functionality. A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

Press **SHIFT + ENTER**. Or press the **Play** button from tool bar to execute the code inside the cell. Wait till kernel becomes idle. 

2. On execution, you will see a confirmation of SparkContext creation.


```
Out[1]: <function __main__.<lambda>>
```

Section 4: Working with data

Import MLib namespace for recommendation

1. Paste the following snippet in an empty cell.

```
from pyspark.sql import *
from pyspark.mllib.recommendation import *
fdir = "/<yourname>/SparkLabDatasets/Lab03/"
salesf = fdir + "SaleTransactions.csv"
```

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

Write a function to generate RDD from data lines

1. Paste the following snippet in an empty cell.

```
def toSalesRec(inline):
    l = inline.split(",")
    return
Row(SaleId=l[0],SessionDateTime=l[1],CustomerAction=l[2],CustomerId=int(l[3]),BookId=int
(l[4]),
              Name=l[5],BookPrice=l[6],Quantity=l[7],
TotalCharges=l[8],Transaction=l[9],OrderId=l[10])
```

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

This function accepts the comma delimited string, splits that string based on comma and converts it into RDD.

Load data from BLOB storage and convert it into RDD

1 - **sc.textFile (string FilePath):** This function accepts file path in string format and loads the data at specified location

2 - **ActionPoints** defines key-value pair where key defines action item and value defines rating

3. Paste the following snippet in an empty cell.

```
print "Loading the Sales Data ..\n"
salesfrdd  = sc.textFile(salesf)
ActionPoints = {"Browsed":3, "Added to Cart":7, "Purchased":10}
salesrdd = salesfrdd.filter(lambda l: l[0] != ',' and l[0].isdigit()).map(lambda l:
toSalesRec(l)).filter(lambda s: s.CustomerAction in ActionPoints.keys())
print "Sales Data loaded..."
```

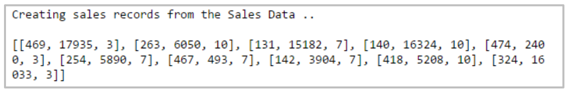
4. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
5. This code snippet loads data from BLOB storage and converts the data lines to stream of RDDs using function defined in previous step.
6. Wait till kernel becomes idle.

Convert the RDD stream to Ratings RDD format

1. Paste the following snippet in an empty cell.

```
print "Creating sales records from the Sales Data ..\n"
ratingsRdd = salesrdd.map(lambda s: [s.CustomerId, s.BookId,
ActionPoints[s.CustomerAction]])
print (repr(ratingsRdd.take(10)))
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. This code snippet converts salesrdd to ratingsRdd having format (UserId,ProductId,rating) required for ALS algorithm.
4. Refer output



```
Creating sales records from the Sales Data ..

[[469, 17935, 3], [263, 6050, 10], [131, 15182, 7], [140, 16324, 10], [474, 240
0, 3], [254, 5890, 7], [467, 493, 7], [142, 3904, 7], [418, 5208, 10], [324, 16
033, 3]]
```

Create trained ALS model

1 - ALS.trainImplicit (ratings, rank, iterations=5, lambda_=0.01, blocks=-1, alpha=0.01, nonnegative=False, seed=None)

ALS.trainImplicit train a matrix factorization model given an RDD of ‘implicit preferences’ given by users to some products, in the form of (userID, productID, preference) pairs. We approximate the ratings matrix as the product of two lower-rank matrices of a given rank (number of features). To solve for these features, we run a given number of iterations of ALS. This is done using a level of parallelism given by blocks.

2 - rank = 10 - means it has a collective rank of higher or equal to 5

3 - numIterations = 5 - means it has occurred atleast 5 times

1. Paste the following snippet in an empty cell.

```
rank = 10
numIterations = 5

print ("Creating an implicit / indirect prediction model (ALS.trainImplicit()) from the
Customer browsing history .. ")
model = ALS.trainImplicit(ratingsRdd, rank, numIterations, alpha=0.02)

savef = '%s/userRecommendationModel' %fdir
model.save(sc, savef)
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. Wait to see the Kernel status turn idle.
4. This code snippet creates trained ALS model using **ALS.trainImplicit** function and saves that model to BLOB storage at location **savef**

Section 5: Recommending products based on Collaborative Filtering

Import Namespaces

1. Paste the following snippet in an empty cell.

```
 from ipywidgets import widgets
 from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
 from IPython.display import display
from pyspark.sql import SQLContext
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. This code snippet import namespaces for getting recommendation, display input box, import python widgets

Write Recommendation Function

1 - **MatrixFactorizationModel.load (sparkContext,path)** loads trained model, by accepting the object of sparkContext and path of location where trained data model is saved in earlier steps.

2 - **recommendedProductIds = sameModel.recommendProducts(customerId, 8)**: This statement gets recommended products from trained model based on provided customerId

3 - **recommendedProducts = recommendedProductsByIds.join(prodnamesRdd)**: This statement joins the **recommendedProductsByIds** RDD with **prodnamesRdd** based on **Product** to get details of product and stores final RDD to **recommendedProducts**

4 - **rpdf = recommendationDetails.toDF()**: This statement converts **recommendationDetails** to dataframe.

5 - **rpdf.write.save (path=savef,source='parquet',mode='overwrite')**: This statement saves the data in dataframe to path **savef** in BLOB storage in **paraquet** format with **overwrite** mode

6 **rpdf.registerTempTable("recommendedProducts")**: This statement creates temporary SQL table **recommendedProducts** based on **rpdf** dataframe and stores data in this table.

1. Paste the following snippet in an empty cell.

```
customerId=int("469")
print(customerId)
loadf = '%s/userRecommendationModel' %fdir
sameModel = MatrixFactorizationModel.load(sc, loadf)
recommendedProductIds = sameModel.recommendProducts(customerId, 8)
print('Recommended product Ids for user ' + str(customerId) +
'\n'.join(map(repr,recommendedProductIds)) + '\n')
prodnames = set(salesrdd.filter(lambda s: s.BookId in [rp[1] for rp in
recommendedProductIds]).map(lambda s: (s.BookId, s.Name)).collect())
recommendedProductsByIds = sc.parallelize(recommendedProductIds).keyBy(lambda b:
b.product)
prodnamesRddRaw = sc.parallelize(prodnames)
prodnamesRdd = prodnamesRddRaw.keyBy(lambda s: s[0])
recommendedProducts = recommendedProductsByIds.join(prodnamesRdd)
recommendationDet=recommendedProducts.map(lambda b:b[1])
recommendationDetails=recommendationDet.map(lambda
b:Row(userid=b[0].user,productid=b[0].product,rating=b[0].rating,productdetail=b[1]))
print(recommendationDetails.collect())
print('Find recommended products for user ' + str(customerId) + ':\n')
print('\n'.join(map(repr,recommendationDetails.collect())) + '\n')
savef = '%s/recommendedProducts' %fdir
print('Saving recommended products to %s ..\n' %savef)
rpdf = recommendationDetails.toDF()
rpdf.write.save(path=savef,source='parquet',mode='overwrite')

rpdf.registerTempTable("recommendedProducts")
sqlContext.sql("select * from recommendedProducts").show()
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. This function gets recommendations from saved trained ALS model.
4. Joins recommendation set with original dataset based on ProductId to get product details of recommended products
5. Final dataset is saved in BLOB storage as paraquet file and temporary SQL table.
6. Try out some Customer IDs for recommendations:

469

263

131

140

474

254

467

142

Section 6: Work with data for FP-Growth

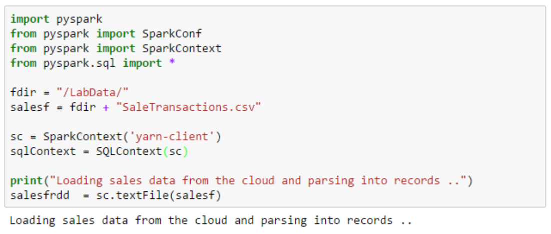
Load data from BLOB storage and convert it into RDD

1 - **sc.textFile (string FilePath)**: This function accepts file path in string format and loads the data at specified location

7. Paste the following snippet in an empty cell.

```
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import *

fdir = "/<yourname>/SparkLabDatasets/Lab03/"
salesf = fdir + "SaleTransactions.csv"

sqlContext = SQLContext(sc)

print("Loading sales data from the cloud and parsing into records ..")
salesfrdd  = sc.textFile(salesf)
```

8. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
9. This code snippet creates SparkContext object and loads data from BLOB storage



```
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import *

fdir = "/LabData/"
salesf = fdir + "SaleTransactions.csv"

sc = SparkContext('yarn-client')
sqlContext = SQLContext(sc)

print("Loading sales data from the cloud and parsing into records ..")
salesfrdd  = sc.textFile(salesf)
```

```
Loading sales data from the cloud and parsing into records ..
```

Write a function to generate RDD from data lines

10. Paste the following snippet in an empty cell.

```
def toSalesRec(inline):
    l = inline.split(",")
    return
Row(SaleId=l[0],SessionDateTime=l[1],CustomerAction=l[2],CustomerId=l[3],BookId=l[4],
               Name=l[5],BookPrice=l[6],Quantity=l[7],
TotalCharges=l[8],Transaction=l[9],OrderId=l[10])
```

11. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
12. This function accepts the comma delimited string, splits that string based on comma and converts it into RDD.

Write a filter function which filters RDD based on CustomerAction and Transaction

13. Paste the following snippet in an empty cell.

```
def sfilter(salesrec):
    return salesrec.CustomerAction in ["Added to cart","Purchased"] and not
salesrec.Transaction in ["Cancelled", "Failed"]
```

14. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
15. This function filters RDD based on CustomerAction and Transaction, allows only successful purchase and 'Added To Cart' transactions.

Convert data lines to filtered RDD stream

16. Paste the following snippet in an empty cell.

```
all_salesrdd = salesfrdd.filter(lambda l: l[0] != ',').map(lambda l: toSalesRec(l))
salesrdd = all_salesrdd.filter(sfilter)
```

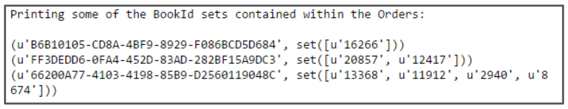
17. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
18. This code snippet removes blank data lines and converts data lines to RDD using function **toSalesRec.**
19. Filters out RDD which denotes failed or Cancelled Purchase transactions using filter function sfilter.

Aggregate RDDs based on OrderId

20. Paste the following snippet in an empty cell.

```
# Group by the orderIds to get the item sets
ordergroups = salesrdd.groupBy(lambda r: r.OrderId)
ordertxns = ordergroups.mapValues(lambda ll: set([l.BookId for l in ll]))
#print out a couple of samples
print("Printing some of the BookId sets contained within the Orders:\n ")
otake = ordertxns.take(3)
print('\n'.join(map(repr,otake)))
```

21. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
22. This code snippet brings the data in (transaction(t),(items (i1),(i2),(i3)..)) format required for FP Growth Algorithm.
23. Output similar to image marks successful execution of code.



```
Printing some of the BookId sets contained within the Orders:

(u'B6B10105-CD8A-4BF9-8929-F086BCD5D684', set([u'16266']))
(u'FF3DEDD6-0FA4-452D-83AD-282BF15A9DC3', set([u'20857', u'12417']))
(u'66200A77-4103-4198-85B9-D2560119048C', set([u'13368', u'11912', u'2940', u'8
674']))
```

Section 7: Train FP-Growth algorithm and find frequent itemsets

Train FP-Growth algorithm

1 - **FPGrowth.train(data, minSupport, numPartitions)**: Computes an FP-Growth model that contains frequent itemsets.

Parameters:

data - The input data set, each element contains a transaction.

minSupport - The minimal support level.

numPartitions - The number of partitions used by parallel FP-growth.

24. Paste the following snippet in an empty cell.

```
from pyspark.mllib.fpm import FPGrowth
minEntries = 3
maxEntries = 10
minSupport = 3
ngroups = ordertxns.count()
minSupportFrac = minSupport /ngroups
model = FPGrowth.train(ordertxns.values(), minSupport=minSupportFrac, numPartitions=10)
```

25. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
26. This code snippet import namespaces for FP-Growth algorithm and prepares the trained model using dataset created in previous section (Section-2)

Create filter function for item entries

27. Paste the following snippet in an empty cell.

```
def ilen(iset):
    return len(iset.items) >= minEntries and len(iset.items) <= maxEntries
```

28. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
29. This code snippet defines a filter which filters the frequent items set to contain entries having item count between **minEntries** and **maxEntries**.

Collect frequent item sets

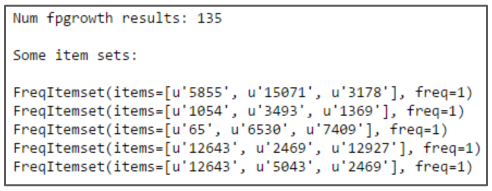
freqItemsets(): Returns the frequent itemsets of this model.

30. Paste the following snippet in an empty cell.

```
fsets = model.freqItemsets().collect()
fsets = filter(ilen, fsets)
print("Num fpgrowth results: " + str(len(fsets)) + '\n')
print("Some item sets: \n")
print('\n'.join(map(lambda f: repr(f), fsets[0:5])))
```

31. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
32. This code snippet collects the frequent itemsets ad filters them using filter **ilen** defined in earlier step,

33. Output of above code resembles with image



```
Num fpgrowth results: 135

Some item sets:

FreqItemset(items=[u'5855', u'15071', u'3178'], freq=1)
FreqItemset(items=[u'1054', u'3493', u'1369'], freq=1)
FreqItemset(items=[u'65', u'6530', u'7409'], freq=1)
FreqItemset(items=[u'12643', u'2469', u'12927'], freq=1)
FreqItemset(items=[u'12643', u'5043', u'2469'], freq=1)
```

Define sql table schema

34. Paste the following snippet in an empty cell.

```
from pyspark.sql.types import *
dataSchema = StructType([StructField("item1", StringType(), True),
                      StructField("item2", StringType(), True),
                      StructField("item3", StringType(), True),
                     StructField("freq", StringType(), True)])
```

35. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
36. This code snippet define table schema for sql table.

Store frequent item sets to Paraquet file and temporary SQL table

1 - **df = rowRdd.toDF()**: This statement converts **rowRdd** to dataframe.

2 - **df.write.save(path=fresults, source='parquet', mode='overwrite')**: This statement saves the data in dataframe to path **fresults** in BLOB storage in **paraquet** format with **overwrite** mode

3 - **df.registerTempTable ("fpgResults")**: This statement creates temporary SQL table **fpgResults** based on **df** dataframe and stores data in this table.

37. Paste the following snippet in an empty cell.

```
rowRdd = sc.parallelize(fsets)
rowRdd1 = rowRdd.map(lambda f: Row(f.items[0],f.items[1],f.items[2],f.freq))
df = sqlContext.createDataFrame(rowRdd1,dataSchema)
fresults = fdir + '/fpgResults'
df.write.save(path=fresults,source='parquet',mode='overwrite')

df.registerTempTable("fpgResults")
```

38. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
39. This code snippet converts RDD to DataFrame and stores data in BLOB storage as Paraquet file and temporary SQL table.

Write event handler

1 - **sqlContext.sql(string query)**: Queries temporary SQL table based on sql query.

40. Paste the following snippet in an empty cell.

```
productId="5855"
print(productId)
query="select count(*) as counter,item1,item2,item3 from fpgResults where
(item1='"+productId+"' OR item2='"+productId+"' OR item3='"+productId+"') AND freq>0
group by item1,item2,item3 order by counter desc"
```

```
sqlContext.sql(query).show()
```

41. Try with other product IDs from the FreqItemSets above
42. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
43. This event handler gets triggered on submit event of inputbox, used to perform query on temporary SQL table to get frequently bought products with product provided by user.

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

Terms of use