Exploring Apache Spark and Spark SQL in Microsoft Azure HDInsight

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

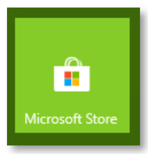
1. Provision an HDInsight Spark Cluster.
2. Access data from Azure storage container and create Dataframe.
3. Understand joins, functions and user defined functions.
4. Connect your HDInsight Spark Cluster with Power BI Visualization.

Prerequisites

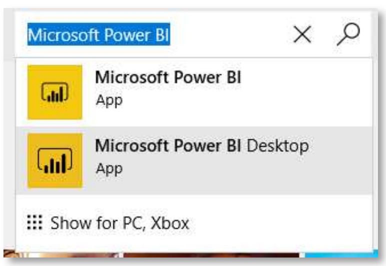
a) An Azure subscription. See here.

a) Microsoft Power BI Desktop See here

1) Launch the Microsoft Store (from windows 10)



2) In the Search bar, type Microsoft Power BI Desktop and select Microsoft Power Bi Desktop.



3) Click on **Install**

Section 1 - Prepare Cluster and dataset

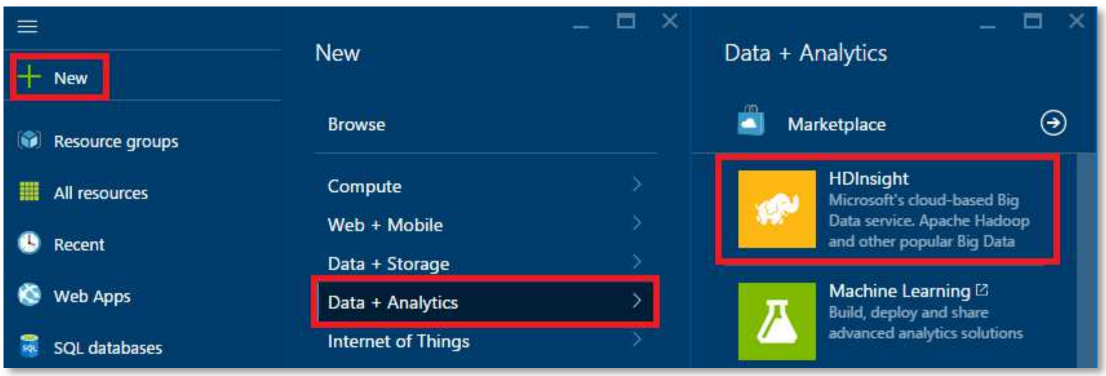
Provision an HDInsight Spark cluster

Access Azure Portal

1. Sign in to the Azure portal.

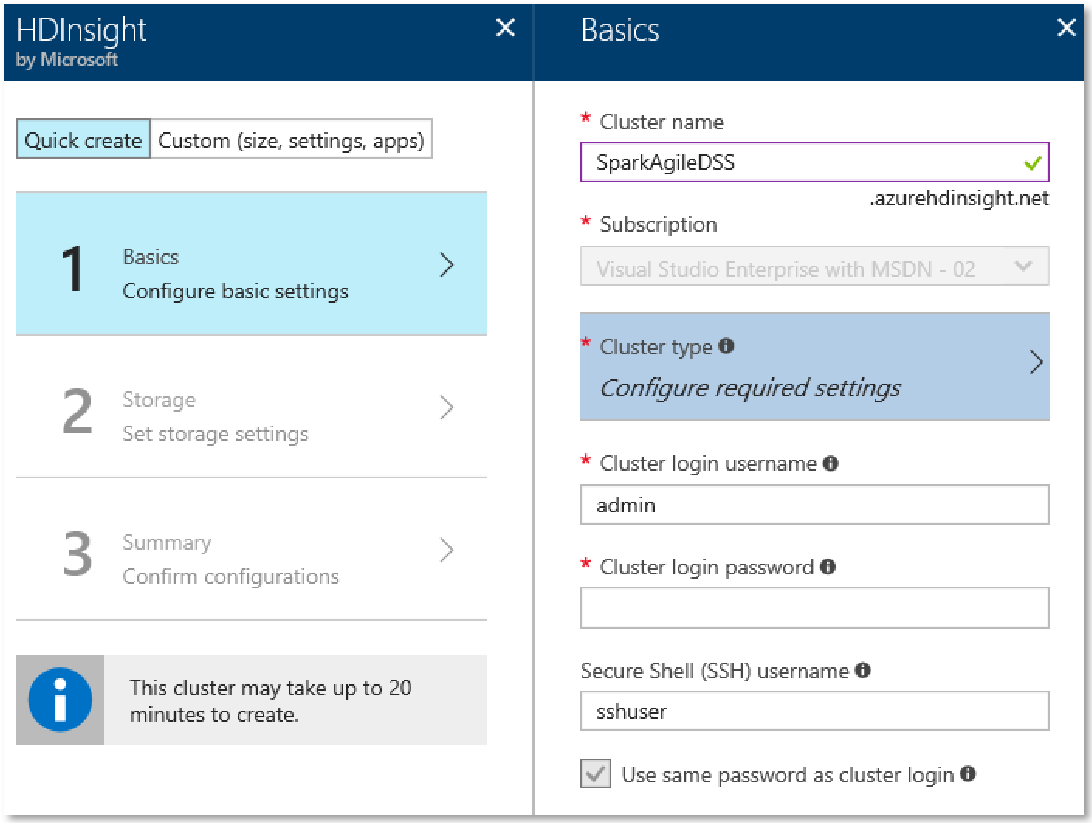
Create HDInsight Spark cluster

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



Provide Cluster Details

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

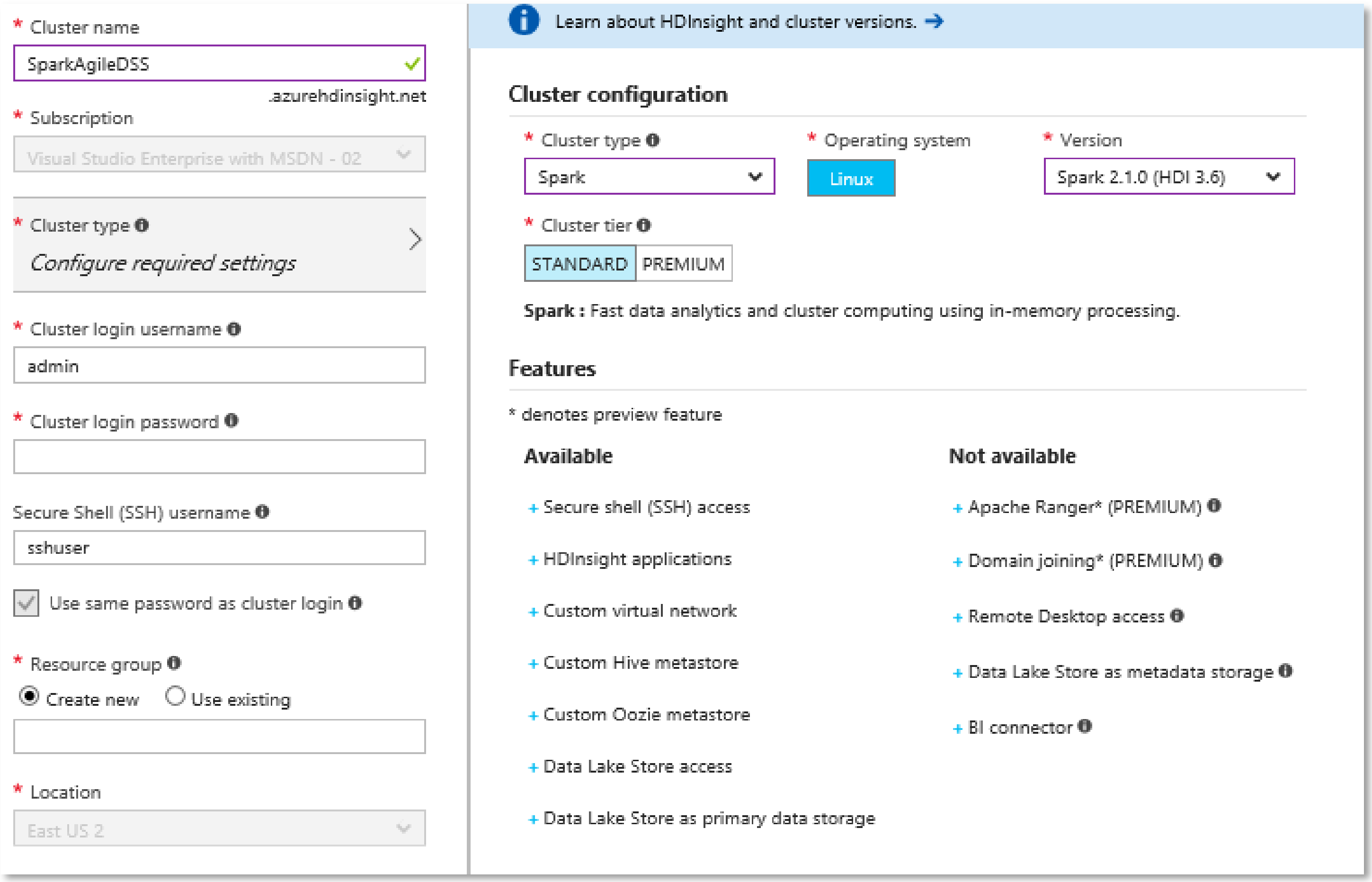


A green check mark appears beside the cluster name if it is available.

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

Configure Cluster Type

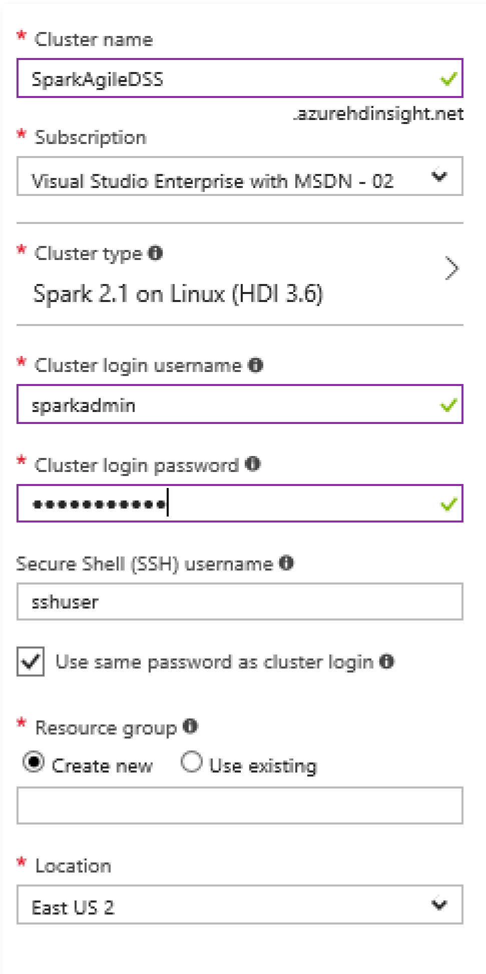
1. Click on **Select Cluster type.** This will open the Cluster Type configuration blade.



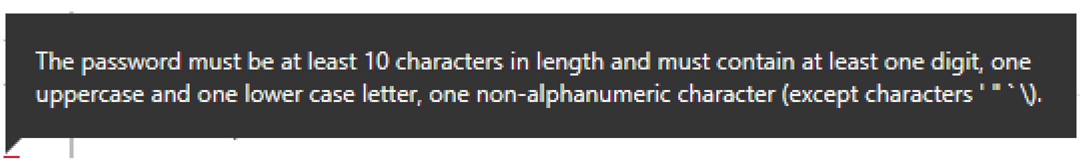
2. Select Spark as Cluster Type.
3. **Operating System** will be **Linux** by default.
4. Select Version as Spark 2.1.0 (HDI 3.6)
5. For Cluster Tier, select STANDARD
6. Click **SELECT** to complete the configuration settings

Provide credentials to access cluster

1. Click Credentials tab to open Cluster Credentials blade.



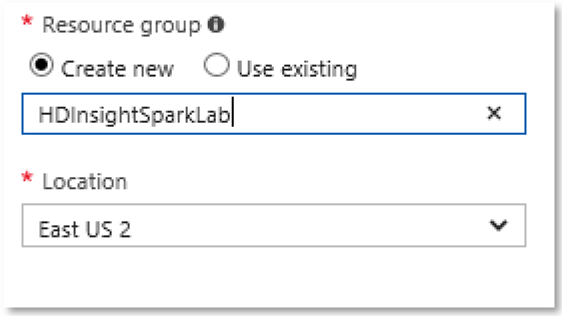
2. Enter Cluster Login Username.
3. Enter Cluster Login Password. Do not forget your username and password, we are going to use it!



4. Enter **SSH Username** (you can leave it to ***sshuser***).
5. Check “Use same password as cluster login”.
6. Click **Select** button to save the credentials.

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

Select Resource group



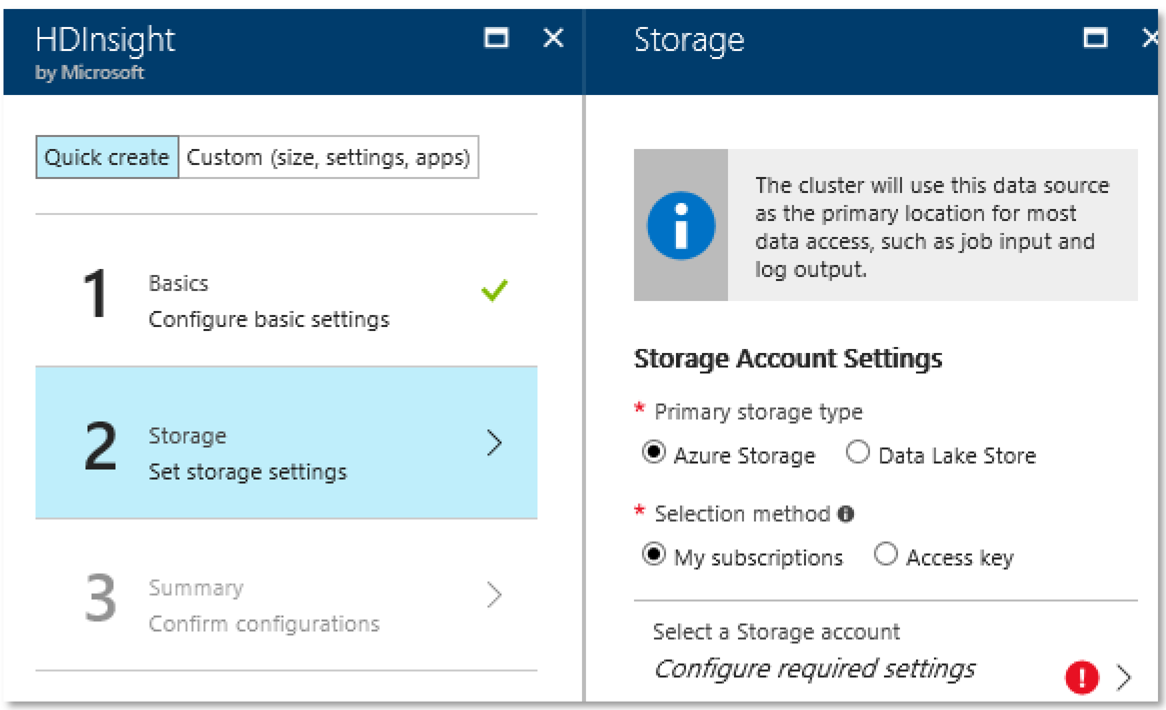
You can specify to create a new Resource group or reuse an existing one. Having this lab under a new Resource Group will facilitate its deletion at the end.

Then click Next to jump to Storage settings.

Storage account settings

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

1. On the **Storage tab,** select **Azure Storage** as Primary storage type



2. **Selection Method** (for the storage account) 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**, else

Option 2 - select **From My Subscriptions** as Selection method.

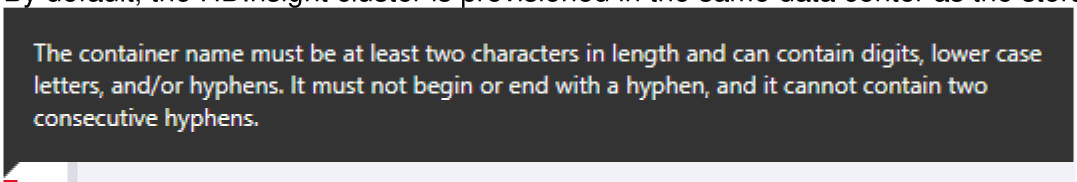
Select **From My Subscriptions** for purpose of this Lab exercise

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 purpose 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**.**

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

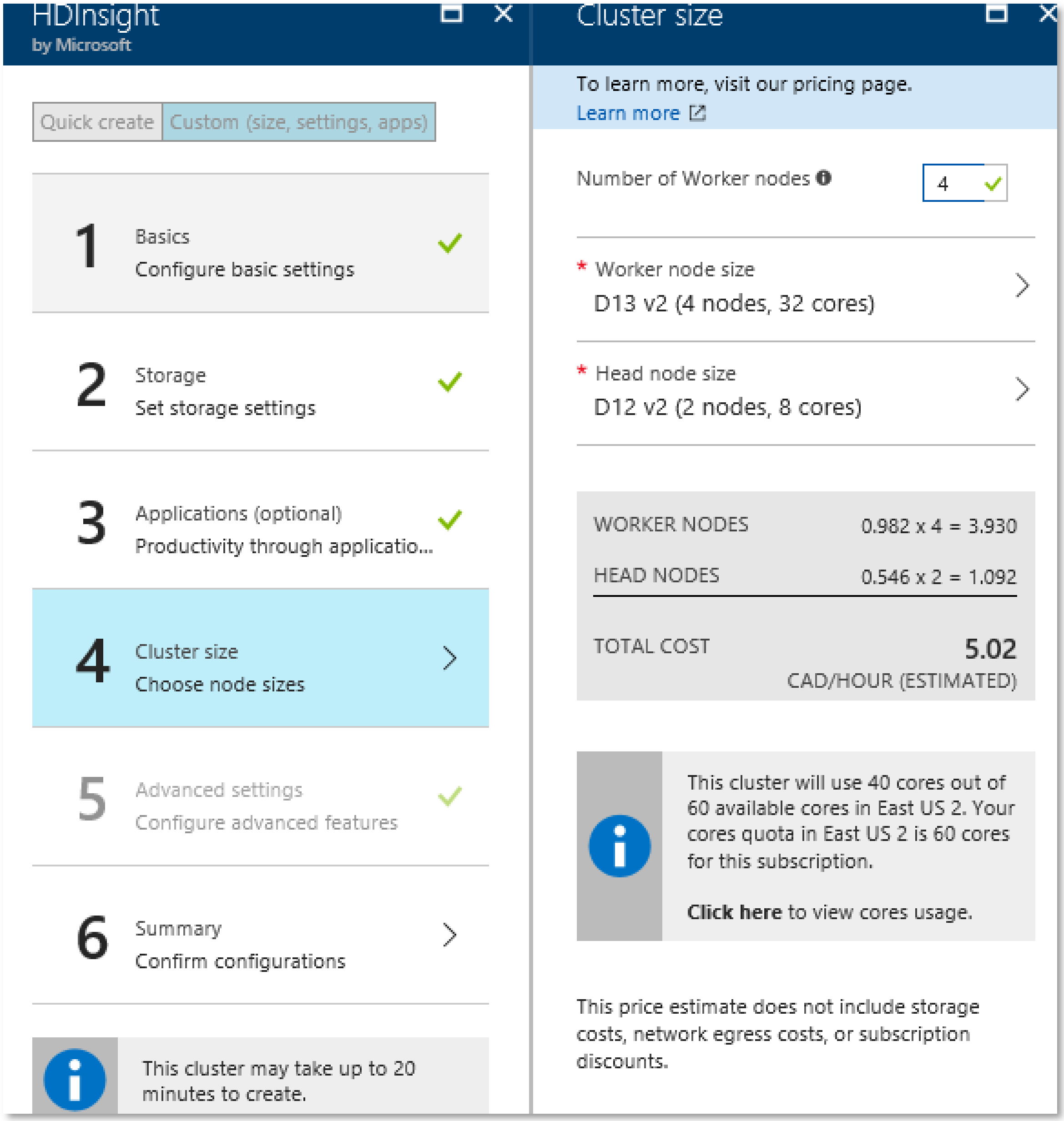


5. Click **Next** button at the bottom to save the data storage configuration.

Summary and Pricing

1. On **Summary tab** we have the ability to edit the **size** of the cluster, by editing the number of nodes: Click **Edit** link near “Cluster size”.

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 $/HOUR.

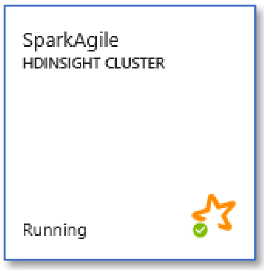
4. Click **Next** button to save node pricing configuration, and leave all other values as default as well.

Provision cluster

1. After completing all the configuration, 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 may take 20 minutes.

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.

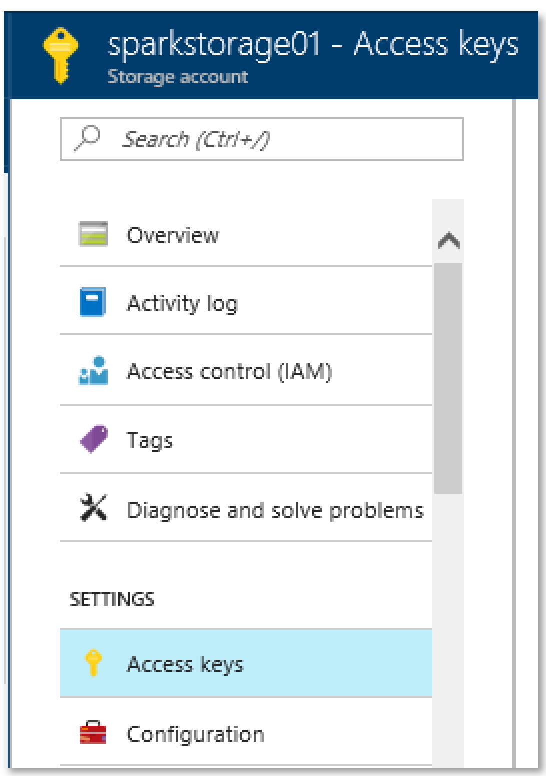


Load datasets files to storage account.

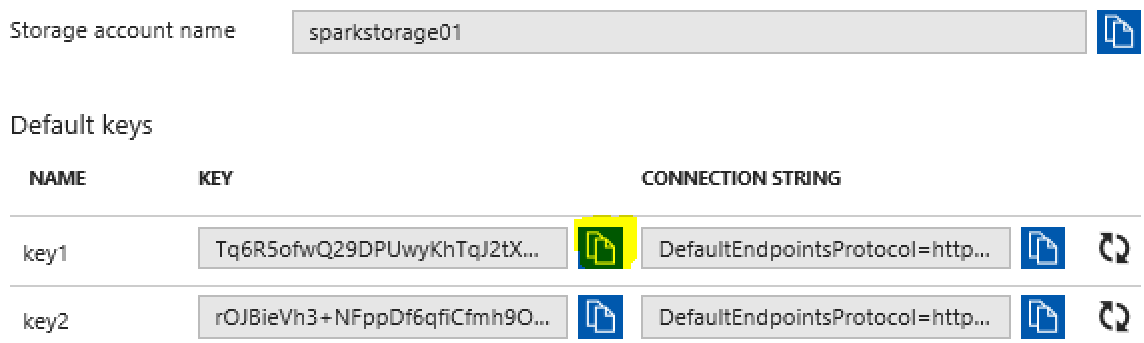
In this section, you'll copy the files required for the lab to the storage account previously created. You'll copy the files between two storage accounts with the help of AzCopy utility. You can download the utility from here http://aka.ms/downloadazcopy

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.



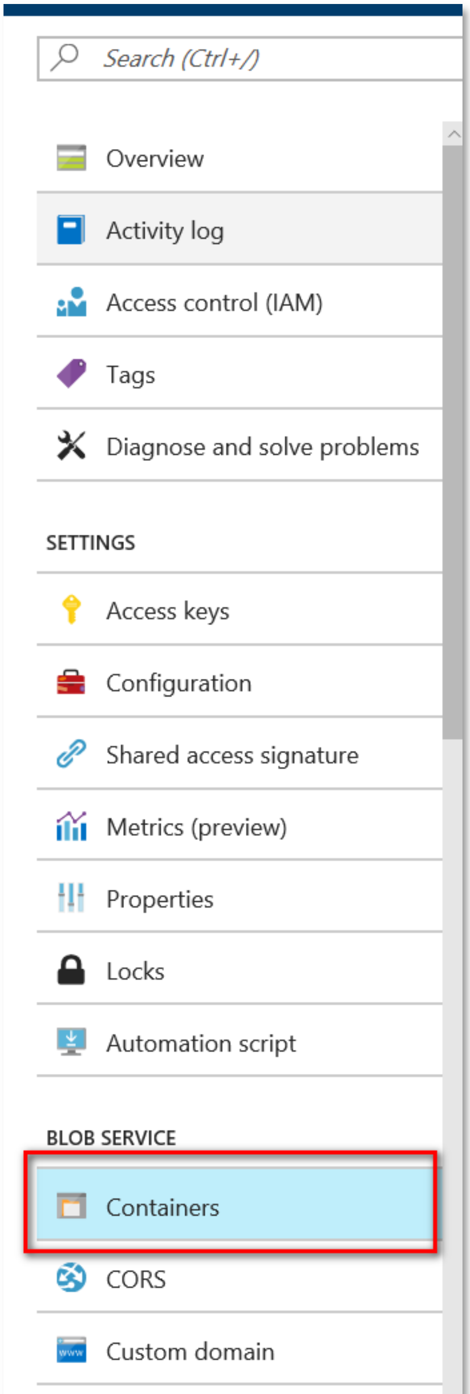
3. Press Window + R to open the run window. Type cmd and press enter to open a new command console window.

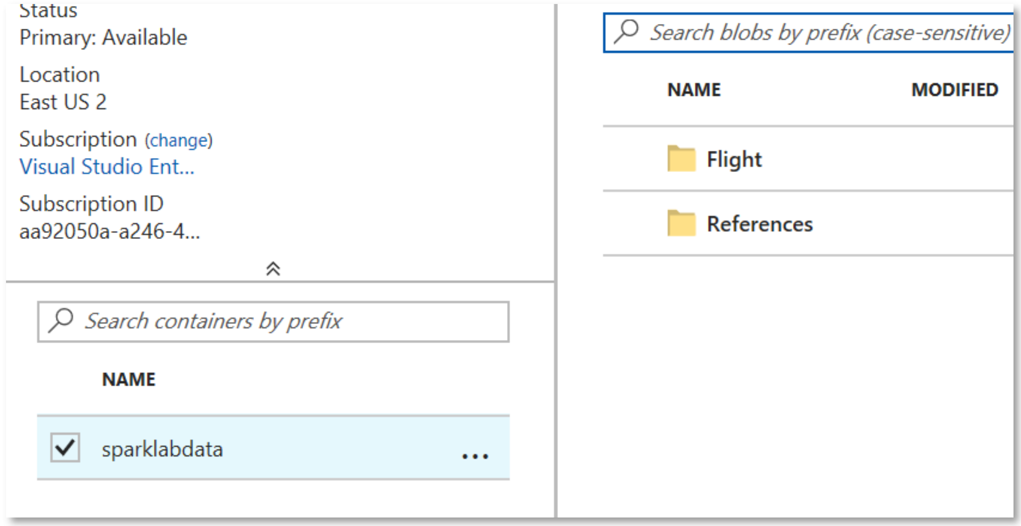
4. Change the directory to C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy.

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. Storage account has been defined during cluster parameterization. “sparklabdata” is the target container inside the storage. If it doesn’t exist it will automatically be created.

```
AzCopy /Source:"https://agilebackup.blob.core.windows.net/sparklabdata/"
/Dest:"https://<your_storage_account_name>.blob.core.windows.net/sparklabdata"
/SourceKey:aqA9yKqg31Dl2Hc29GGU+bbGCBi6qBKKlN320dvXqgDX9HZIuPfaiWMe3arNlEZqrMZSBQzd
qGIclUCpLo+Fyg== /DestKey:<your_dest_key> /S
```

6. In Azure portal, navigate to your storage account, then Containers below ‘BLOB SERVICE’ (see following screenshot), and verify that a new container ‘sparklabdata’ has been created, containing all the resources:


Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
SETTINGS
Access keys
Configuration
Shared access signature
Metrics (preview)
Properties
Locks
Automation script
BLOB SERVICE
Containers
CORS
Custom domain

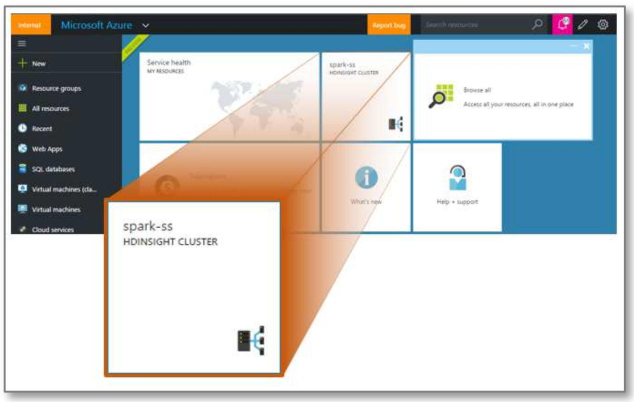

Status
Primary: Available
Location
East US 2
Subscription (change)
Visual Studio Ent...
Subscription ID
aa92050a-a246-4...
Search containers by prefix
NAME
sparklabdata
Search blobs by prefix (case-sensitive)
NAME
MODIFIED
Flight
References

Section 2 - Spark SQL and Dataframe

Access data from Azure storage container and Create Data frame.

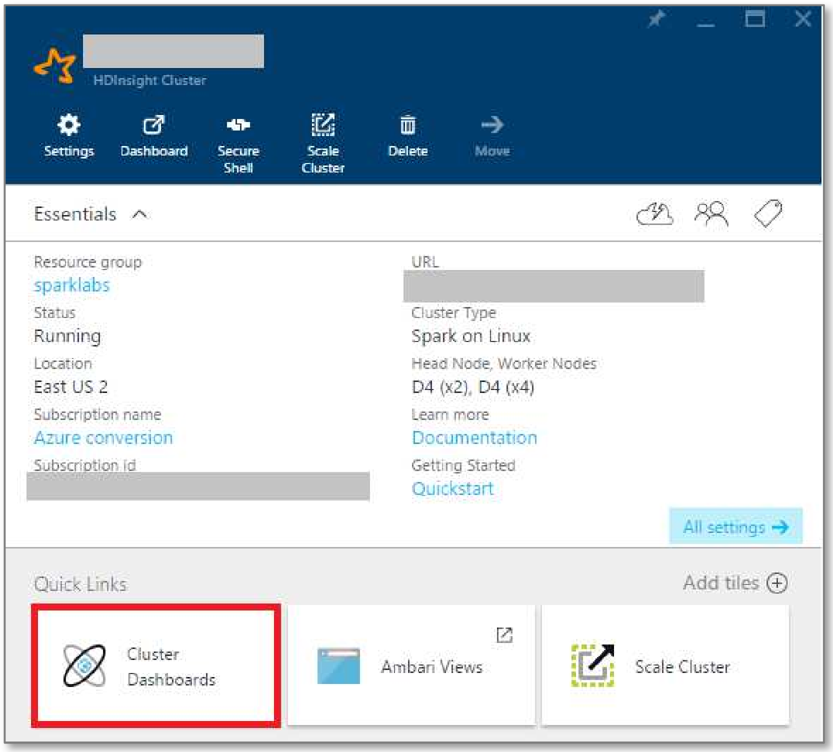
Access Azure

1. Sign in to the **Azure portal**.
2. Click tile for your Spark Cluster.

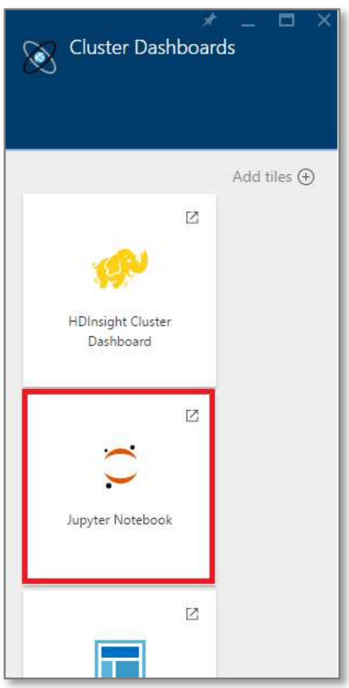


Launch Jupyter Notebook

1. Click on **Cluster Dashboards** tile present on the Cluster Blade.



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

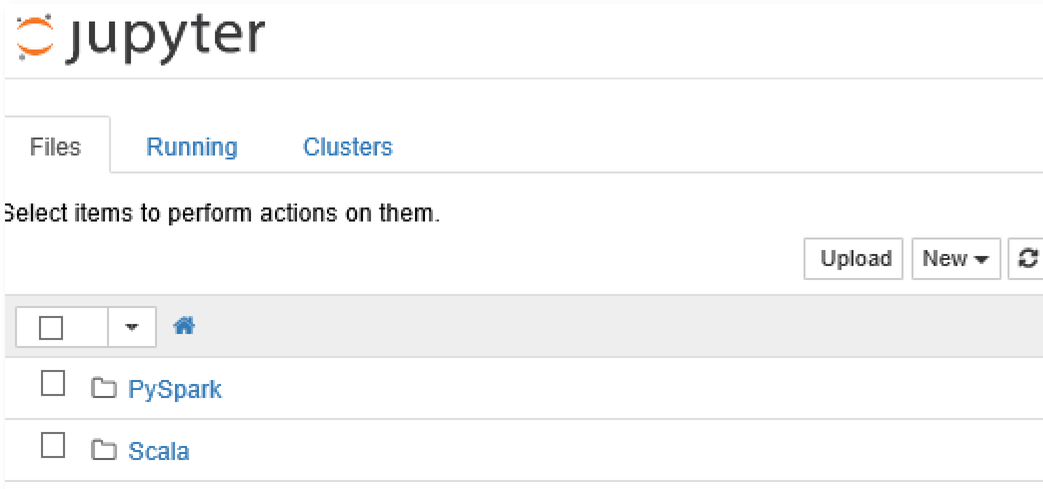


When prompted, use the admin credential of your Spark Cluster.

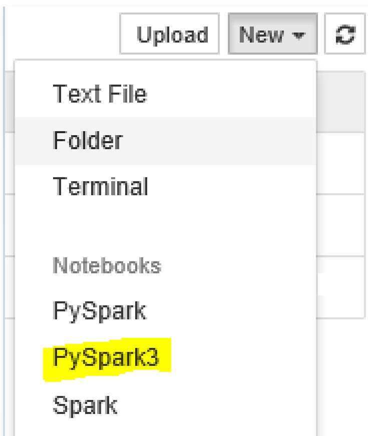
Create a new Jupyter Notebook

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

Jupyter Notebook will open.



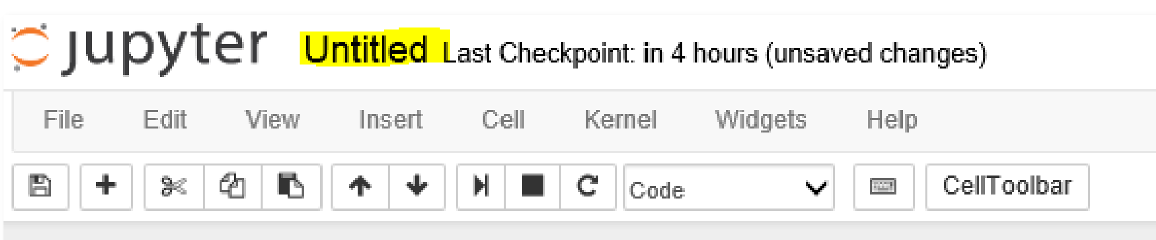
1. Click **New** dropdown button at top right side of Jupyter Notebook screen.
2. Click **PySpark3** under Notebooks, from the dropdown.



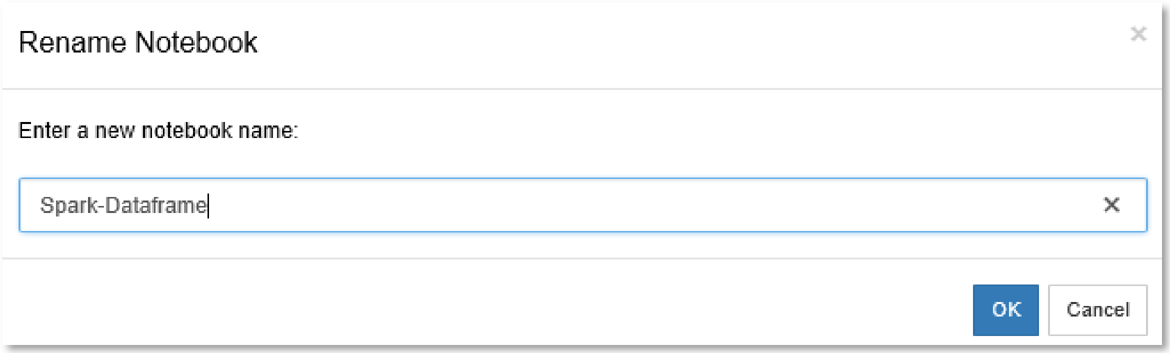
Assign friendly name to notebook

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 a new name and press button **OK**.

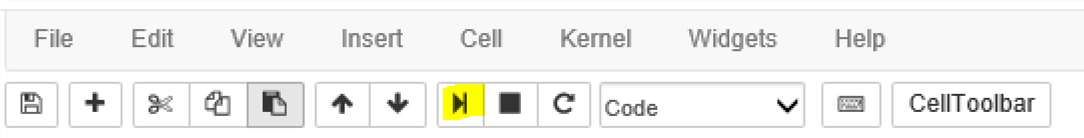


Create Spark and SQL context

Starting from Spark2.0 there is no need to import and start SparkContext and SQLContext!

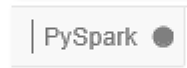
Create data frame from data stored in azure blob storage

To a bloc of code you can click on the arrow at the top:

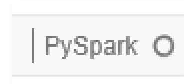


Every time you run a job in Jupyter, your web browser window title will show a (Busy) status alongside the notebook title.

You will also see a solid circle next to the PySpark text in the top-right corner.



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



1. Paste the following snippet in below empty cell, do not forget to replace <container_name> (should be sparklabdata) and <storage_account_name>.

```
# Define dataset azure path
flightPerfFilePath
="wasb://<container_name>@<storage_account_name>.blob.core.windows.net/Flight/*/*.c
sv"

# Obtain dataframe
flightPerf =
spark.read.format("com.databricks.spark.csv").options(header='true').load(flightPer
fFilePath)
```

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

3. Output of above code execution will be as shown below, meaning Spark application correctly started:

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
1	application_1506710033677_0005	pyspark3	idle	Link	Link	✔

```
SparkSession available as 'spark'.
```

4. Verify “flightPerf” data type, it should be “DataFrame”. You can paste this code in an empty cell and run it:

```
type(flightPerf)
```

```
<class 'pyspark.sql.dataframe.DataFrame'>
```

```
#try flightPerf alone
flightPerf
```

DataFrame operations, explore the data

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

1. Total number or rows:

```
flightPerf.count()
```

Output:

```
20563827
```

2. Look at the data structure:

```
flightPerf.printSchema()
```

Output:

```
root
 |-- YEAR: string (nullable = true)
 |-- QUARTER: string (nullable = true)
 |-- MONTH: string (nullable = true)
 |-- DAY_OF_MONTH: string (nullable = true)
 |-- DAY_OF_WEEK: string (nullable = true)
 |-- FL_DATE: string (nullable = true)
 |-- UNIQUE_CARRIER: string (nullable = true)
 |-- AIRLINE_ID: string (nullable = true)
 |-- CARRIER: string (nullable = true)
 |-- TAIL_NUM: string (nullable = true)
 |-- FL_NUM: string (nullable = true)
 |-- ORIGIN_AIRPORT_ID: string (nullable = true)
 |-- ORIGIN_AIRPORT_SEQ_ID: string (nullable = true)
 |-- ORIGIN_CITY_MARKET_ID: string (nullable = true)
 |-- ORIGIN: string (nullable = true)
 |-- ORIGIN_CITY_NAME: string (nullable = true)
 |-- ORIGIN_STATE_ABR: string (nullable = true)
 |-- ORIGIN_STATE_FIPS: string (nullable = true)
 |-- ORIGIN_STATE_NM: string (nullable = true)
 |-- ORIGIN_WAC: string (nullable = true)
 |-- DEST_AIRPORT_ID: string (nullable = true)
 |-- DEST_AIRPORT_SEQ_ID: string (nullable = true)
 |-- DEST_CITY_MARKET_ID: string (nullable = true)
 |-- DEST: string (nullable = true)
 |-- DEST_CITY_NAME: string (nullable = true)
 |-- DEST_STATE_ABR: string (nullable = true)
 |-- DEST_STATE_FIPS: string (nullable = true)
 |-- DEST_STATE_NM: string (nullable = true)
 |-- DEST_WAC: string (nullable = true)
 |-- CRS_DEP_TIME: string (nullable = true)
 |-- DEP_TIME: string (nullable = true)
 |-- DEP_DELAY: string (nullable = true)
 |-- DEP_DELAY_NEW: string (nullable = true)
```

…

We can see that our dataset has quite a lot of columns!

3. flightPerf.show()

   Output… not very readable with our dataset…

```
|YEAR|QUARTER|MONTH|DAY_OF_MONTH|DAY_OF_WEEK|   FL_DATE|UNIQUE_CARRIER|AIRLINE_ID|CARRIER|TAIL_NUM|FL_NUM|ORIGIN_AI
RPORT_ID|ORIGIN_AIRPORT_SEQ_ID|ORIGIN_CITY_MARKET_ID|ORIGIN|    ORIGIN_CITY_NAME|ORIGIN_STATE_ABR|ORIGIN_STATE_FIPS
|ORIGIN_STATE_NM|ORIGIN_WAC|DEST_AIRPORT_ID|DEST_AIRPORT_SEQ_ID|DEST_CITY_MARKET_ID|DEST|     DEST_CITY_NAME|DEST_S
TATE_ABR|DEST_STATE_FIPS| DEST_STATE_NM|DEST_WAC|CRS_DEP_TIME|DEP_TIME|DEP_DELAY|DEP_DELAY_NEW|DEP_DEL15|DEP_DELAY_
GROUP|DEP_TIME_BLK|TAXI_OUT|WHEELS_OFF|WHEELS_ON|TAXI_IN|CRS_ARR_TIME|ARR_TIME|ARR_DELAY|ARR_DELAY_NEW|ARR_DEL15|AR
R_DELAY_GROUP|ARR_TIME_BLK|CANCELLED|CANCELLATION_CODE|DIVERTED|CRS_ELAPSED_TIME|ACTUAL_ELAPSED_TIME|AIR_TIME|FLIGH
```

4. We can select specific columns:

```
flightPerf.select("AIRLINE_ID","FL_DATE").show()
```

Output:

```
+----------+----------+
|AIRLINE_ID|   FL_DATE|
+----------+----------+
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
|     20355|2014-01-16|
+----------+----------+
only showing top 20 rows
```

5. Apply some filter and show only 1 row:

```
flightPerf.filter(flightPerf.AIRLINE_ID == 19805).select("Origin","Dest").show(1)
```

```
+------+----+
|Origin|Dest|
+------+----+
|   JFK| LAX|
+------+----+
only showing top 1 row
```

6. We can also rename the output columns:

```
flightPerf.filter(flightPerf.AIRLINE_ID == 19805)\
        .select(flightPerf.ORIGIN.alias('FROM'),flightPerf.DEST.alias('TO'))\
        .show(1)
```

Running SQL Queries

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

```
flightPerf.registerTempTable("flightPerfTable")
```

2. Execute below SQL querie in empty cell and observe the output

```
%%sql
SELECT
YEAR,UNIQUE_CARRIER,AIRLINE_ID,ORIGIN,DEST,DEP_DELAY_NEW,CANCELLED,CANCELLATION_COD
E,AIR_TIME,DISTANCE
FROM flightPerfTable
```

Output:

AIRLINE_ID	AIR_TIME	CANCELLATION_CODE	CANCELLED	DEP_DELAY_NEW	DEST	DISTANCE	ORIGIN	UNIQUE_CARRIER	YEAR
20355	314	NaN	0	8	SFO	2296	CLT	US	2014-01-01
20355	51	NaN	0	0	DCA	331	CLT	US	2014-01-01
20355	133	NaN	0	0	FLL	899	DCA	US	2014-01-01
20355	104	NaN	0	0	MSY	651	CLT	US	2014-01-01
20355	75	NaN	0	0	EWR	529	CLT	US	2014-01-01
20355	110	NaN	0	7	CLT	936	DFW	US	2014-01-01
20355	77	NaN	0	0	CLT	449	PHL	US	2014-01-01
20355	52	NaN	0	27	BWI	361	CLT	US	2014-01-01

3. Let’s find out how many rows we have by year:

```
%%sql
SELECT YEAR,count(*) as NbrFlights
FROM flightPerfTable GROUP BY YEAR
```

Output:

YEAR	NbrFlights
2016-01-01	5617658
2017-01-01	3307279
2014-01-01	5819811
2015-01-01	5819079

4. Verify that the counts are similar here:

```
flightPerf.groupBy("YEAR").count().sort("YEAR").show()
```

Notice that 2017 has significantly less flights, and it makes sense because the year is not over yet. But what is our last month?

Perform operations on data frames to analyze the data

Use some analytic functions

Some useful functions:

- **groupBy(*cols)**: Groups the DataFrame using specified columns, in order to run aggregation on them.
- **count():** Returns the number of rows in DataFrame.
- **collect():** Returns all records as list of row.
- **orderBy(*cols, ascending=True/False):** Returns a new DataFrame sorted by the specified columns.
- **avg(*args):** Computes average values for each numeric column for each group.
- **sum(*args):** Computes sum for each numeric column for each group.

1. Get the number of arrival flights by state in 2016

```
flightPerf.filter(flightPerf.YEAR == 2016).groupBy("DEST_STATE_NM").count().show()
```

Output:

```
+--------------------+------+
|       DEST_STATE_NM| count|
+--------------------+------+
|                Utah|111559|
|              Hawaii|100815|
| U.S. Virgin Islands|  6117|
|           Minnesota|135827|
|U.S. Pacific Trus...|   489|
|                Ohio| 69059|
|              Oregon| 68263|
|            Arkansas| 16179|
|               Texas|578440|
|        North Dakota| 12739|
|        Pennsylvania|107598|
|         Connecticut| 20322|
|            Nebraska| 21349|
|             Vermont|  4044|
|              Nevada|165517|
|         Puerto Rico| 29606|
|          Washington|147083|
|            Illinois|340426|
|            Oklahoma| 30829|
|              Alaska| 36144|
+--------------------+------+
only showing top 20 rows
```

2. **Try by yourself**: Select top 5 States from previous output

3. **Try by yourself**: For those 5 states, calculate the number of flights variation (in %), year over year (from 2014 to 2015, and 2015 to 2016).

Here is the desired output:

Year	State	NbrFlights	prevYear_Var
2015-01-01	Illinois	418264	6.745476
2015-01-01	Florida	449248	4.950754
2016-01-01	California	727407	2.775651
2015-01-01	Georgia	394412	2.165511
2016-01-01	Georgia	398518	1.041043
2016-01-01	Florida	447168	-0.462996
2015-01-01	Texas	688031	-4.142849
2015-01-01	California	707762	-4.249090
2016-01-01	Texas	578440	-15.928207
2016-01-01	Illinois	340426	-18.609778

There are multiple ways of achieving this, for example:

- A first step could be to generate a new Dataframe, with only the subset of data that we need.
- Then we register our dataframe as a temp table,
- Then we can finalize our analysis in SQL, giving us more flexibility with advance functions (window functions for example)

Learn how to JOIN dataset

1. Show top 5 origin cities having the most flight cancellation

```
flightPerf.filter(flightPerf.CANCELLED == 1)\
.groupBy("ORIGIN_CITY_NAME")\
.count()\
.orderBy("count",ascending=False)\
.show(5)
```

Output:

```
+--------------------+-----+
|    ORIGIN_CITY_NAME|count|
+--------------------+-----+
|         Chicago, IL|35390|
|        New York, NY|20779|
|Dallas/Fort Worth...|17050|
|         Atlanta, GA|13440|
|          Newark, NJ|12912|
+--------------------+-----+
only showing top 5 rows
```

2. Find out the main reasons of the cancellations

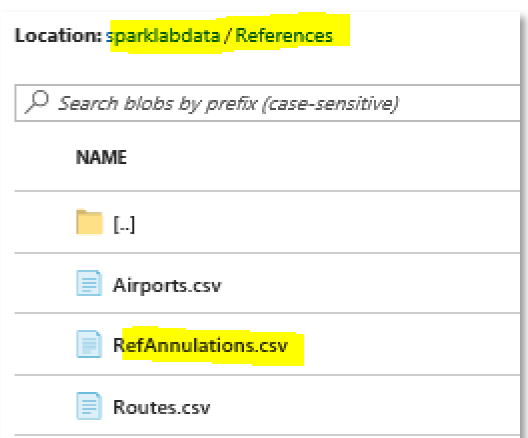
Our dataset only has the CANCELLATION_CODE, if we want to show the count by cancellation description, we need to use another dataset, “RefAnnulations.csv”, and join it to our current dataframe.

- Firstly, generate a dataframe with our flights count by cancellation code:

```
flightCancel = flightPerf.filter(flightPerf.CANCELLED == 1)\
.groupBy("CANCELLATION_CODE")\
.count()
```

- **Try by yourself**: Generate a new dataframe “flightCancelRef“ from the file RefAnnulations.csv

Here is the location of the reference file:



- **Try by yourself**: Now we have everything we need in order to join our dataframes together. Here is the synthax:

DataframeA.join(DataframeB, DataframeA.key == DataframeB.key)

How does the output look like?

Data type conversion and statistical functions

One of the main advantage of PySpark/Scala over SQL is the access to a ton of libraries, for statistical purpose and matrix calculation for example.

1. As a simple example, calculate the correlation coefficient between the AIR_TIME and DISTANCE. For that we can use the function “corr”, taking in arguments 2 columns of a dataframe (using the Pearson method).

   - Let’s try this:

```
flightPerf.stat.corr("AIR_TIME","DISTANCE")
```

Output:

```
'requirement failed: Currently correlation calculation for columns with dataType StringType not suppor
ted.'
Traceback (most recent call last):
  File "/usr/hdp/current/spark2-client/python/pyspark/sql/dataframe.py", line 1654, in corr
    return self.df.corr(col1, col2, method)
  File "/usr/hdp/current/spark2-client/python/pyspark/sql/dataframe.py", line 1425, in corr
    return self._jdf.stat().corr(col1, col2, method)
  File "/usr/hdp/current/spark2-client/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py", line 113
3, in __call__
```

Oops… corr function is based on numeric values, and it looks like Spark is not automatically converting our strings into numeric values.

   - Manually cast the data and assign result into a new dataframe:

```
DF =
flightPerf.select(flightPerf.AIR_TIME.cast('float'),flightPerf.DISTANCE.cast('float
'))
```

   - Try again the correlation calculation

```
DF.stat.corr("AIR_TIME","DISTANCE")
```

Output:

```
0.9795776861248402
```

Nearly perfect correlation (coefficient is always between -1 and 1), but you already probably guessed it, as this correlation is quite obvious...

Visualize the results

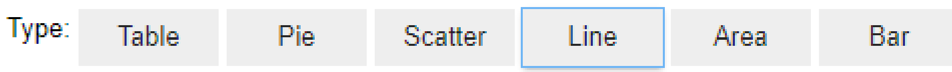
Try by yourself: Find out the State destination with the bigger difference in 2016, in term of number of flights, between 2 months (variation in %).

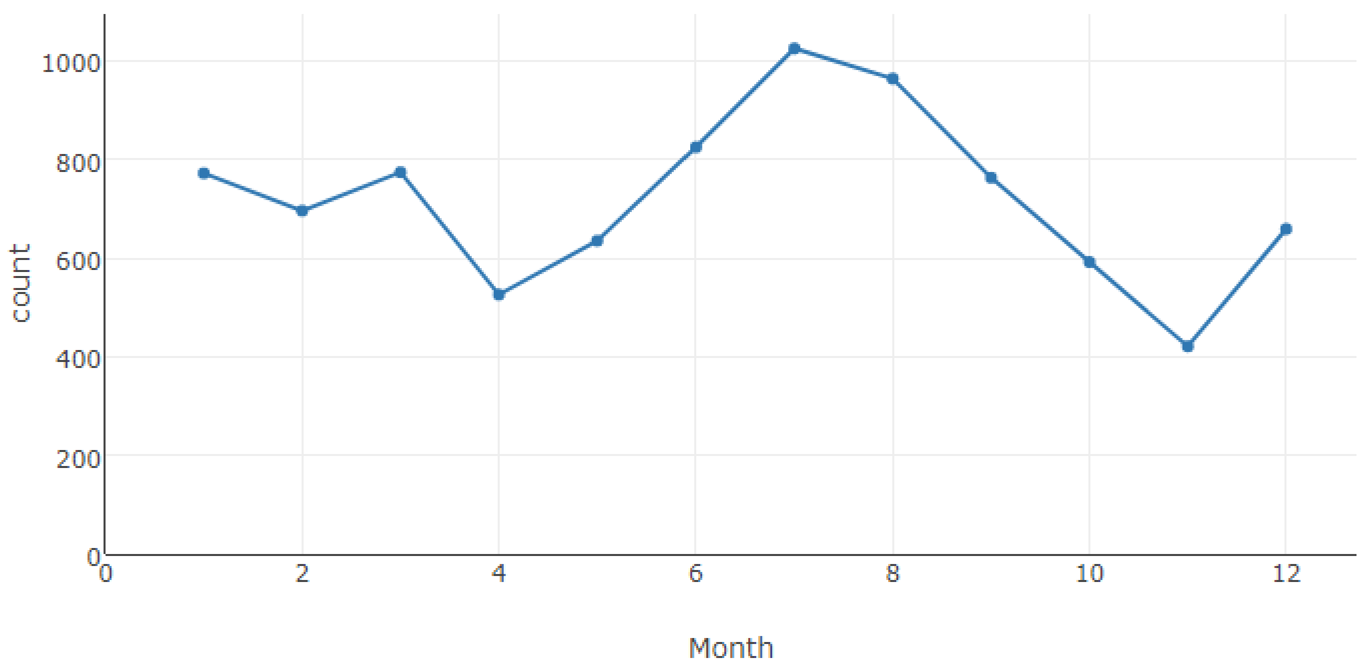
- To resolve this, first you can build a temp table containing the count of flights by MONTH / DEST_STATE_NAME
- From here you can calculate the variation in %

1. Our winner should be Wyoming. Let's visualize the month trend for this state (here we assume that a temp table "flightMonthTable" has been created, containing the count of flights by MONTH, YEAR and DEST_STATE_NM):

```
%%sql
SELECT
CAST(MONTH as INTEGER)
AS Month,count
FROM flightMonthTable
WHERE YEAR = 2016 AND DEST_STATE_NM = 'Wyoming'
ORDER BY 1
```

You can easily change the output at the top, a line chart for example:





We can see that Wyoming receives more flights during the summer.

Section 3 - Power BI on Spark HDInsight

To Design a Power BI report based on Spark, we need to persist our data into a Hive table.

Dataframe to HIVE

1. Open a new Jupyter notebook
2. Create the hive table with this data: the number of flights and average delay (DEP_DELAY) by destination state for each departure city. To be more representative we will only consider the flights having a delay > 1 hour.

    - Create the initial DataFrame

    ```
    # Flight dataset path
    flightFilePath
    ="wasb://<container_name>@<storage_account_name>.blob.core.windows.net/Flight/*/*.c
    sv"
    # Dataframe
    departureDelays =
    spark.read.format("com.databricks.spark.csv").options(header='true').load(flightFil
    ePath)
    ```

    - Build our query and assign it to a new dataframe

    ```
    # Register a temp table
    departureDelays.registerTempTable("departureDelays")

    # New dataframe
    AvgDelay = spark.sql("SELECT  ORIGIN_CITY_NAME as OriginCity, DEST_STATE_NM as
    DestinationState, 'United States' as Country, AVG(DEP_DELAY) as AverageDelay,
    COUNT(*) as DelayFrequency FROM departureDelays WHERE DEP_DELAY > 60 GROUP BY
    ORIGIN_CITY_NAME, DEST_STATE_NM ")

    # Register final temp table
    AvgDelay.createOrReplaceTempView("avgDelay")
    ```

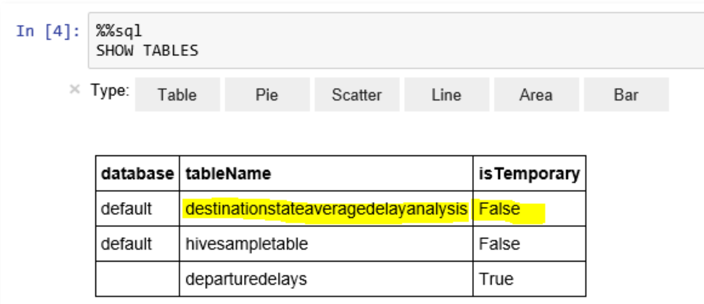
    - Create Hive Table

    ```
    spark.sql("create table DestinationStateAverageDelayAnalysis as select * from
    avgDelay ")
    ```

3. Let’s check out where the hive table has been created.

```
%%sql
SHOW TABLE
```

Please compare your result

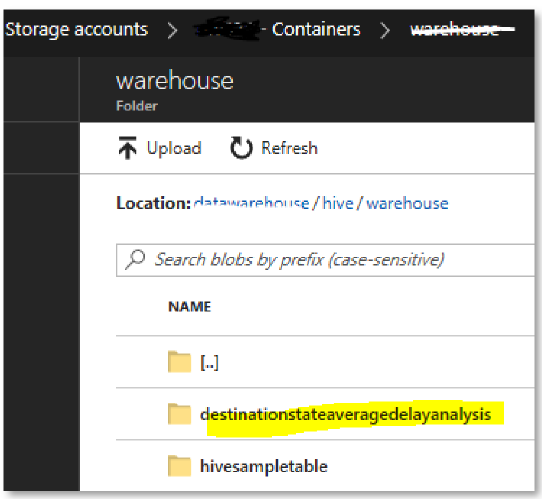


In [4]:
```
%%sql
SHOW TABLES
```

Type: Table Pie Scatter Line Area Bar

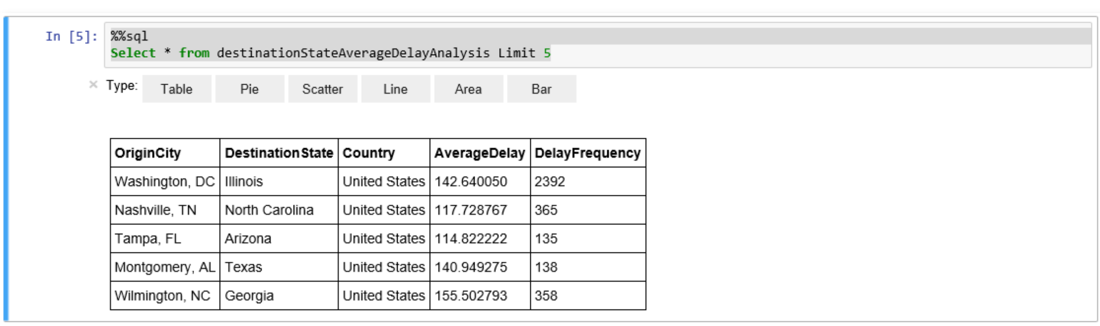
database	tableName	isTemporary
default	destinationstateaveragedelayanalysis	False
default	hivesampletable	False
	departuredelays	True

4. To understand where your table and your data were saved, please go-back on the azure-portal Tab in your browser,
5. drill-through your **Storage accounts**,
6. explore your **cluster container** and expand the **hive**/**warehouse** folder
7. What did you observe?



8. Now go back on the notebook tab, and type the following command to query your table

```
%%sql
Select * from destinationStateAverageDelayAnalysis Limit 5
```



In [5]:
```
%%sql
Select * from destinationStateAverageDelayAnalysis Limit 5
```

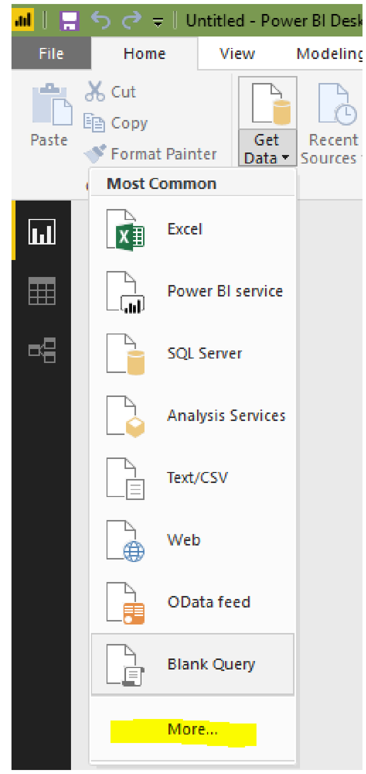
Type: Table Pie Scatter Line Area Bar

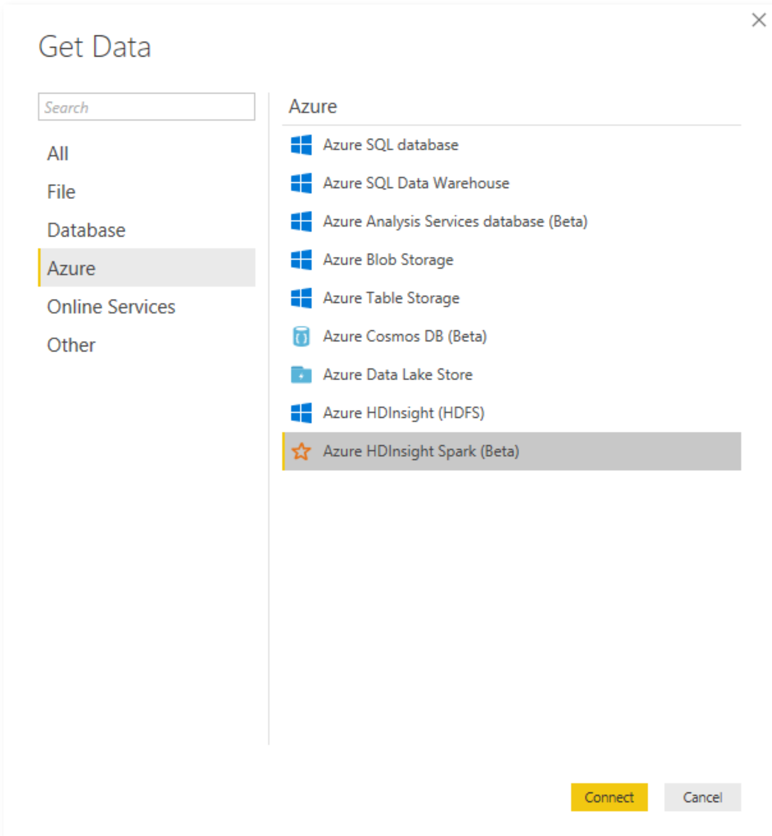
OriginCity	DestinationState	Country	AverageDelay	DelayFrequency
Washington, DC	Illinois	United States	142.640050	2392
Nashville, TN	North Carolina	United States	117.728767	365
Tampa, FL	Arizona	United States	114.822222	135
Montgomery, AL	Texas	United States	140.949275	138
Wilmington, NC	Georgia	United States	155.502793	358

Connect an Azure HDInsight Spark Datasource

In this exercise, you'll connect Power BI to the previous hive table.

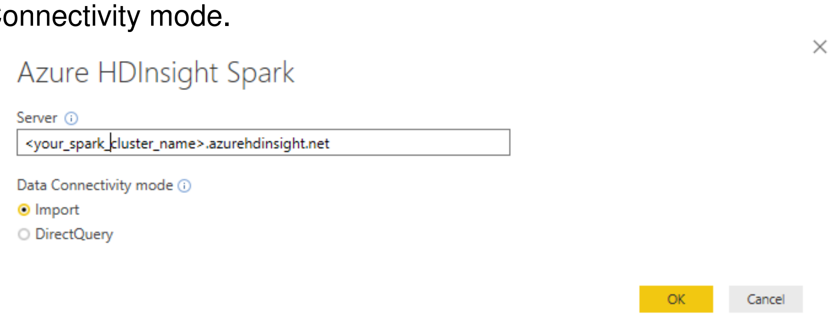
1. Open you Microsoft Power BI Desktop application
2. With a new report, inside the **Home** tab, expand the **Get Datasource** menu and select the **More…** option
3. In the **Get Data** dialog window, on the left side, select **Azure**.



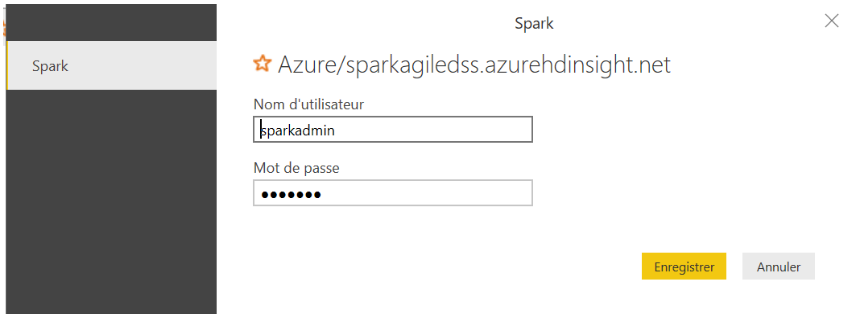


4. Click "**Connect**"
5. In the Form **Azure HDInsight Spark** window, in the Server input box, enter the cluster url:

Server: <your_cluster_name>.azurehdinsight.net, it's important to check the **Import** option in the Data Connectivity mode.

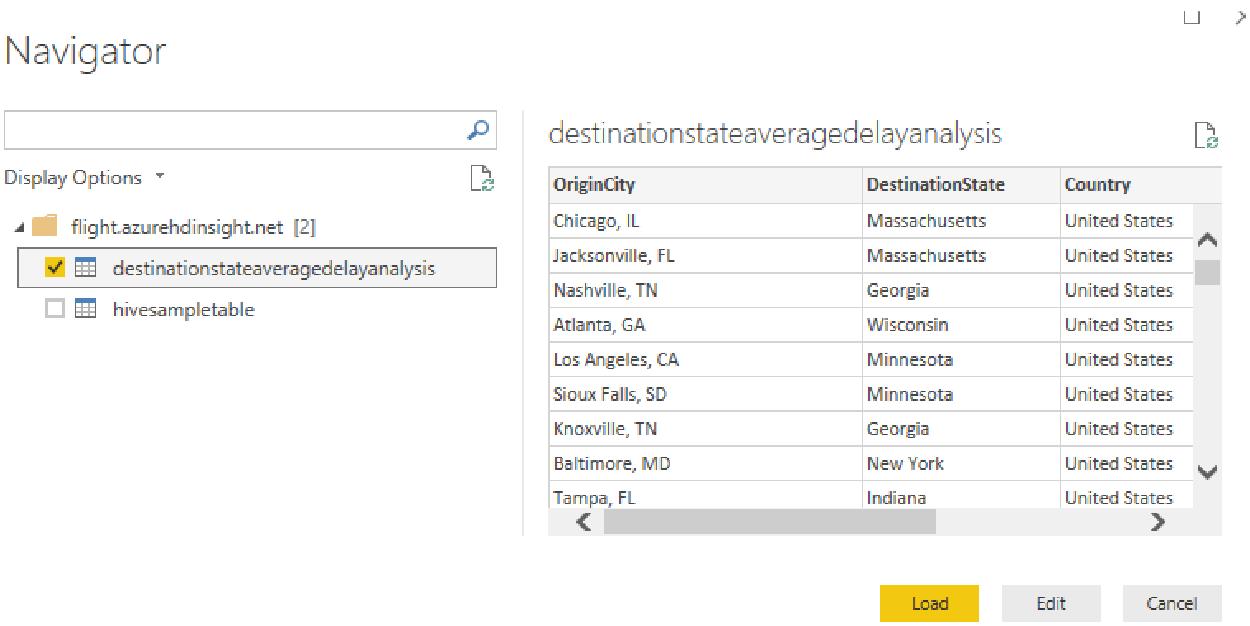


6. Click "**OK**"
7. Register your cluster's credentials



8. Click “**Connect**”.

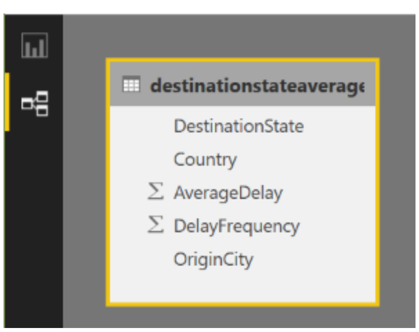
9. In the Navigator dialog window, expand the HIVE database, and then expand **<your_cluster_name>**.azurehdinsight.net
10. Check the following table.



11. Click **Load**.
12. Explore your data model in the diagram tab at the left.

The data will be loaded into the Power BI Desktop file.

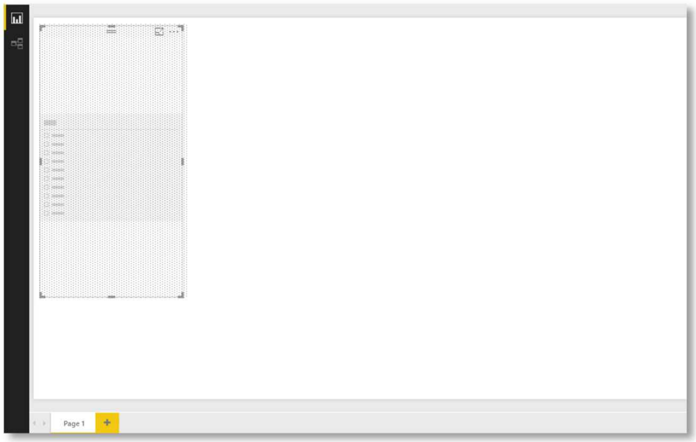
Once loaded, in the **Queries** pane (located at the left), select the query to review the data from the Hive table.



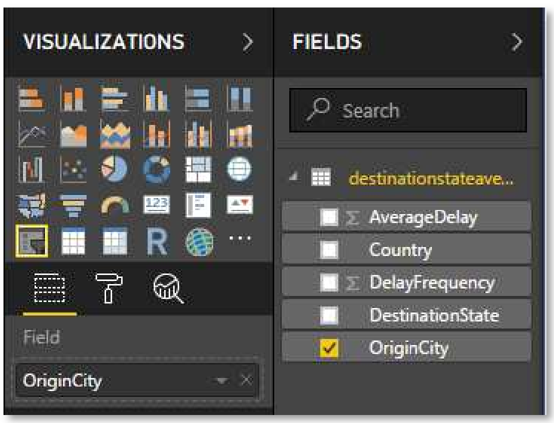
Designing the Power BI report

In this exercise, you will design an interactive report based on the hive table.

1. Go to the report pane
2. To add a Segment from inside the Visualization pane, click the **Slicer** icon
3. Reposition and resize the visualization based on the following diagram.

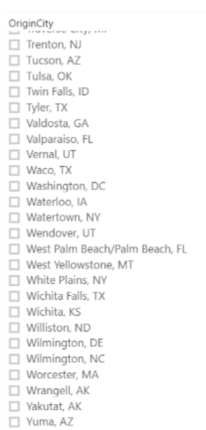


4. In the Fields pane (located at the right), Expand the **destinationStateAverageDelayAnalysis** table.



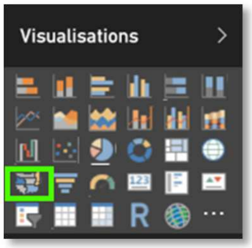
5. From the Fields pane, inside the expanded table, check the **OriginCity** field.

Verify that the visualization looks like the following



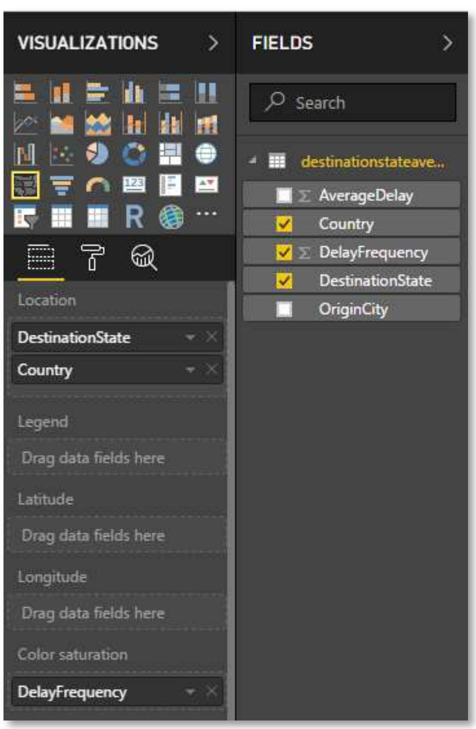
6. To add a Map, from inside the Visualization pane, click on the **Filled Map** icon.

Tips: you can hover the cursor over each icon to reveal a tooltip describing the type of visualization.



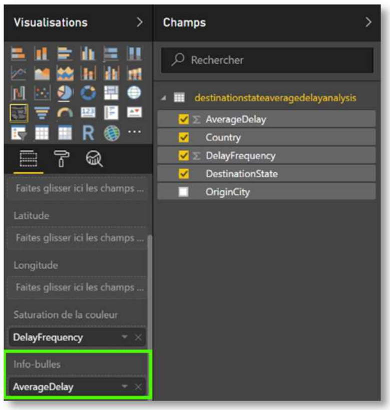
7. Reposition and resize the map visualization based on the following diagram.

8. From the Fields pane, inside the expanded table, drag the **DestinationState** to Emplacement property and repeat the operation with the **Country** bellow the **DestinationState** .

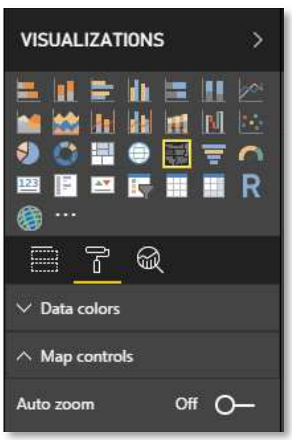


9. From the Fields pane, inside the expanded table, drag the **DelayFrequency**, to the **Color saturation** property.

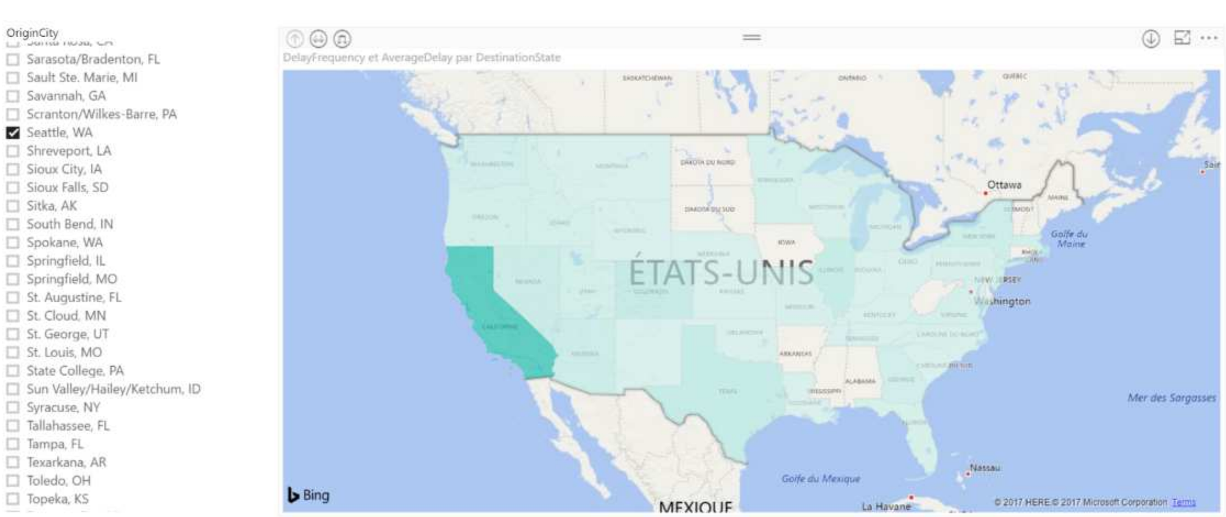
10. From the Fields pane, from inside the expanded table, drag the **AverageDelay**, to the Tool Tips property



11. To disable the **Auto Zoom** feature on the Map, on the format icon, expand the Map command, and turn **off** the **Auto zoom** property



12. Verify that the visualization looks like the following



RDD & User Define Function

In this final exercise you will create a new Hive table, and connect a Power BI visualization on it to display the traffic flow. We will use this exercise to introduce to you the RDD API and the user define functions.

1. Create a new notebook, copy paste the following statement
2. Redefine the Data filepath of the source

```
# Obtain airports and flights dataset

AirportFilePath = "wasb:// <container_name>@<storage_account_name>.blob.core.windows.net/References/Airports.csv"

FlightFilePath = "wasb:// <container_name>@<storage_account_name>.blob.core.windows.net/Flight/*/*.csv"
```

3. Instantiate and create a cleansing function in python to parse the Airports reference files
    a. Remove, quote and double quote, and trim each value.

```
from pyspark.sql.types import *

#function quote remover and Trim
from pyspark.sql.functions import udf

def clean(x):
    for i in range(len(x)):
      x[i]=x[i].replace('"','').replace('\'',' ').strip()
    return(x)
```

4. Load a new RDD, apply some transformations on the Airport file.
    a. Get only airports from the United States

```
# RDD creation
# split document in lines
airportData = sc.textFile(AirportFilePath)
USAirportDataFinal = airportData.map(lambda l: l.split(",")).map(clean).filter(lambda c: c[3] == 'United States' and c[4] != '\\N')
```

5. Apply a new Schema definition on the RDD

```
airportDataFields =  [StructField("AirportId", StringType(), True),
                    StructField("Name", StringType(), True),
                    StructField("City", StringType(), True),
                    StructField("Country", StringType(), True),
                    StructField("IATA", StringType(), True),
                    StructField("ICAO", StringType(), True),
                    StructField("Latitude", StringType(), True),
                    StructField("Longitude", StringType(), True),
                    StructField("Altitude", StringType(), True),
                    StructField("Timezone", StringType(), True),
                    StructField("DST", StringType(), True),
                    StructField("TzDatabase", StringType(), True),
                    StructField("Type", StringType(), True),
                    StructField("Source", StringType(), True)]

# Apply schema to the RDD
airportDataSchema = StructType(airportDataFields)
```

6. Load the Flight dataset into a new DataFrame

```
#Creation du DataFrame depuis le RDD
airportData_DataFrame = USAirportDataFinal.toDF(airportDataSchema)
flight_df = sqlContext.read.format("com.databricks.spark.csv").options(header='true').load(FlightFilePath)
```

7. Create temporary view based on the two DataFrames

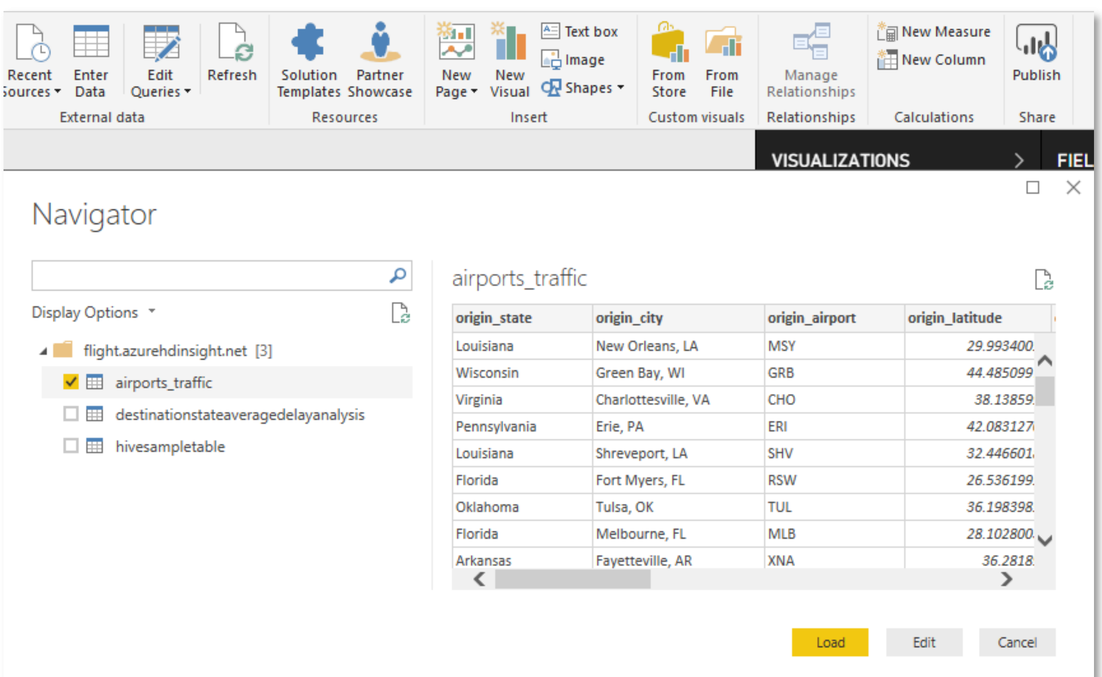
```
## Creates a temporary view based on the DataFrame
airportData_DataFrame.createOrReplaceTempView("airports_na")
flight_df.createOrReplaceTempView("departureDelays")
```

8. Do the projection of Flights with the enrichment of the Latitude and Longitude of each Airport's location.

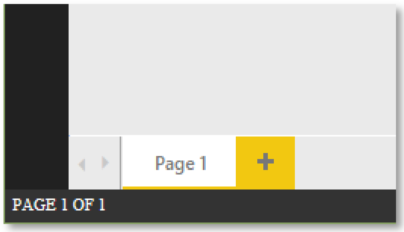
```
airport_traffic = sqlContext.sql("SELECT \
                        ORIGIN_STATE_NM as origin_state, \
                        ORIGIN_CITY_NAME as origin_city, \
                        ORIGIN as origin_airport, \
                        cast(O.Latitude as double) as origin_latitude,\
                        cast(O.Longitude as double) as origin_longitude, \
                        DEST_STATE_NM as destination_state, \
                        DEST_CITY_NAME as destination_city, \
                        DEST as destination_airport, \
                        cast(Dest.Latitude as double) as dest_latitude, \
                        cast(Dest.Longitude as double) as dest_longitude, \
                        COUNT(*) as FlightCount, AVG(DEP_DELAY) as dep_delay, \
                        AVG(ARR_DELAY)  s arr_delay \
                      FROM departureDelays D \
                      JOIN airports_na O  ON D.ORIGIN = O.IATA \
                      JOIN airports_na Dest  ON D.DEST = Dest.IATA \
                      GROUP BY ORIGIN_STATE_NM, ORIGIN_CITY_NAME, ORIGIN, O.Latitude, O.Longitude, DEST_CITY_NAME, DEST,
DEST_STATE_NM, Dest.Latitude, Dest.Longitude ")

airport_traffic.write.saveAsTable('airports_traffic')
```

9. Return on the Microsoft Power BI Desktop and click on the **Recent Sources** icon in the **Home** ribbon.

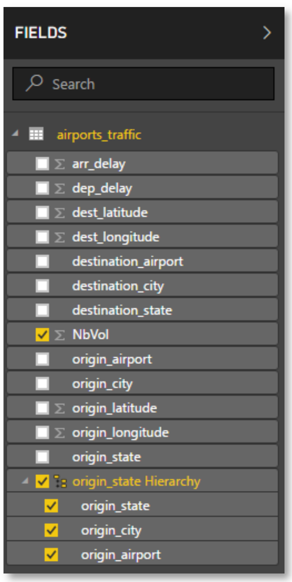
10. Select **spark clustername** sources, check the new **airports_traffic**, and push the **Load** button.



11. On your report you can observe the new table in the Fields panel named **airport_traffic**, add a **new page** in the bottom of the report and click on **+**



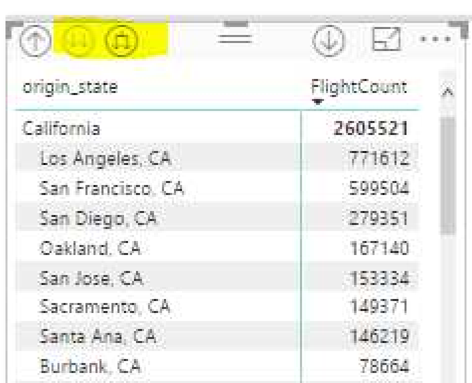
12. On your page 2, refactor your airports_traffic field panel: create a new hierarchy, drag and drop the **origin_city** on the **origin_state**, a new field named **origin_state Hierarchy** will be created, continue and add the **origin_airport** by drag and drop.

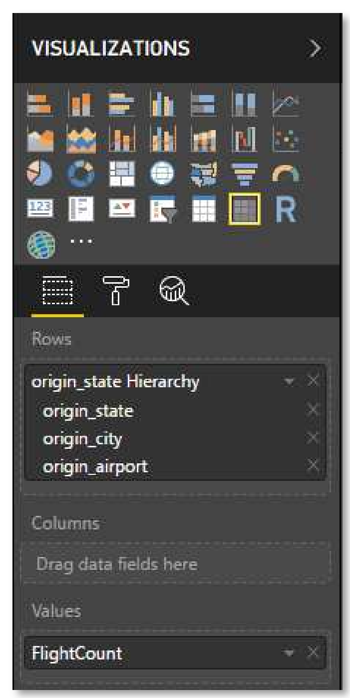


13. Add a matrice visualization, and add the origin_state_hierarchy as row and FlightCount as Value

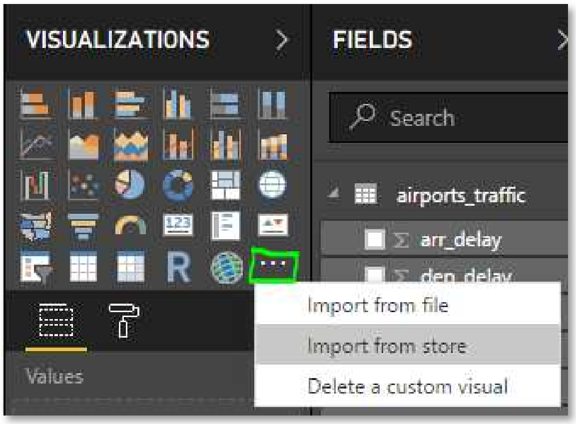
    a. Sort the matrice by **FlightCount** decreasing

    b. Tips: you can expand the next level or only the next level on selected item, click on the **FlightCount** column to sort by the highest number of flight.



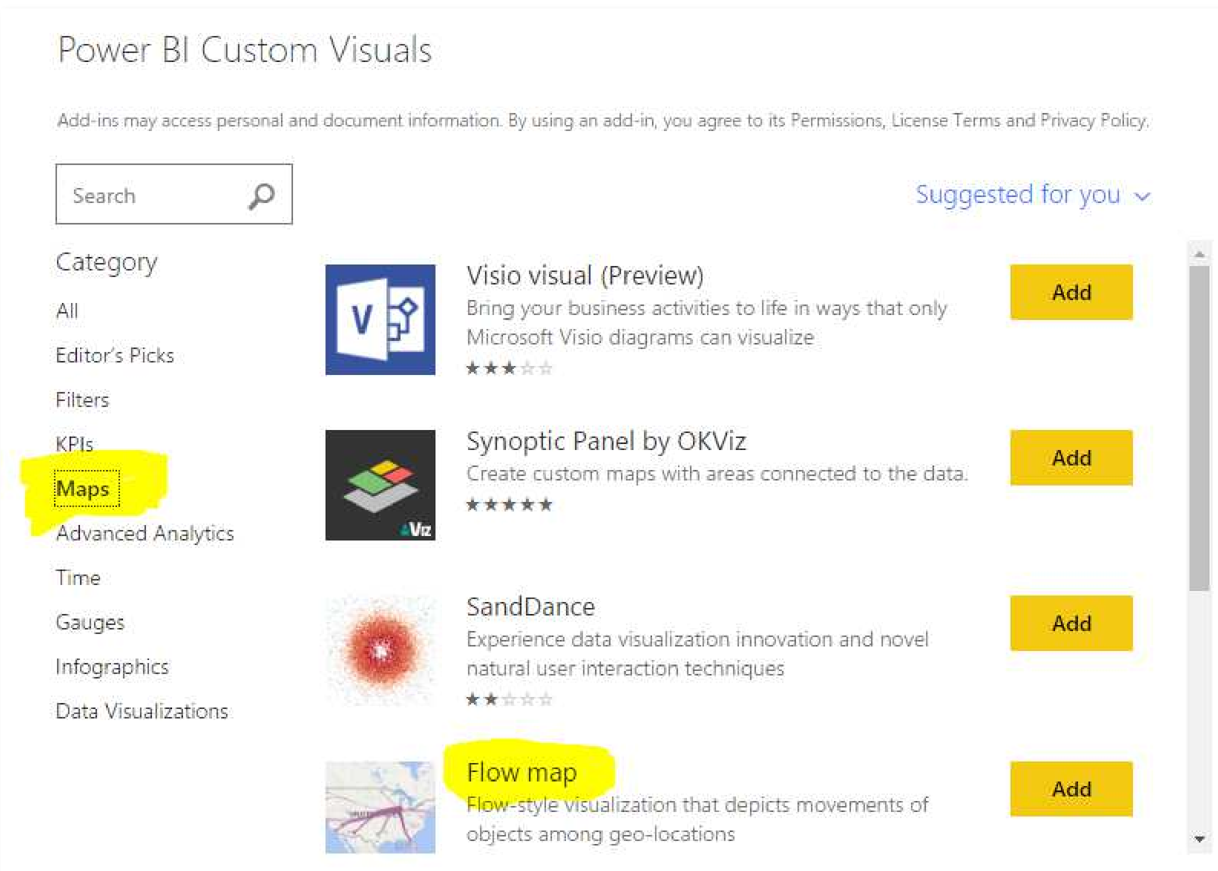


14. Add a new visualization from the store :



15. Select the … and select Import from store.

16. When the Power BI Custom Visuals Store open, select the **Maps** category and choose the **Flow map** and Add.



17. Add on the **map flow properties** and **place fields** as the snapshot

   a. Drag & drop the field **origin_airport** to the Map flow's **Origin** property

   b. Drag & drop the field **destination_airport** to the Map flow's **Destination** property

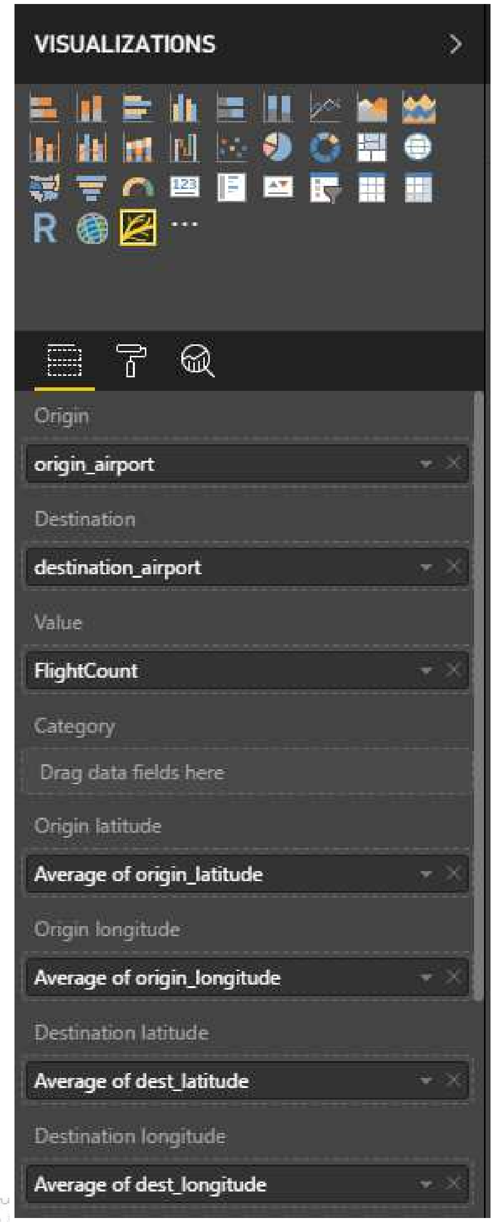
   c. Drag & drop the field **flightCount** to the Map flow's **Value** property

   d. Drag & drop the field average of **origin_latitude** to **Origin latitude**

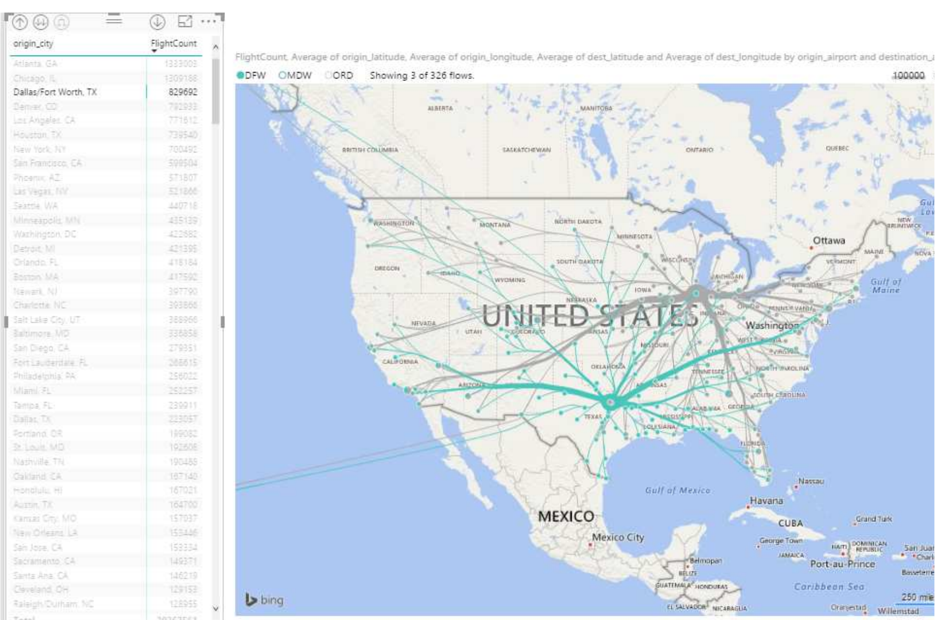
   e. Drag & drop the field average of **origin_longitude** to **Origin longitude**

   f. Drag & drop the field average of **dest_latitude** to **Destination latitude**

   g. Drag & drop the field average of **dest_longitude** to **Destination longitude**



18. Select an **origin_city** in the **matrice**. You should have something similar to this:



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

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 agileDSS 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.