



Lab Manual- Azure Data Bricks Provisioning and Data Ingestion Part1

Prepared for:

Date: 18th March 2022

Prepared by:

Document Name: Lab Manual

Document Number AZLabn990

Contributor:

Contents

1.	Introduction.....	3
2.	Exercise 1 – Ingest Data from GitHub.....	3
3.	Exercise: Run code in the 1-Delta-Architecture notebook.....	5
1.	Datasets Used	6
2.	Define Schema.....	6
3.	Define Streaming Load from Files in Blob	7
4.	WRITE Stream using Delta Lake.....	7
5.	Load Static Lookup Table	7
6.	Create QUERY tables (aka "silver tables")	8
7.	Create QUERY tables (aka "silver tables")	8
8.	See list of active streams.	9
9.	Gold Table: Grouped Count of Events.....	9
10.	Gold Table: Grouped Count of Events.....	10
11.	Materialized View: Windowed Count of Hourly gt Events.....	10

1. Introduction

Systems are working with massive amounts of data in petabytes or even more and it is still growing at an exponential rate. Big data is present everywhere around us and comes in from different sources like social media sites, sales, customer data, transactional data, etc

[Apache Spark](#) is an open-source, fast cluster computing system and a highly popular framework for big data analysis. This framework processes the data in parallel that helps to boost the performance. It is written in [Scala](#), a high-level language, and also supports APIs for Python, SQL, Java and R.

Azure Databricks is the implementation of Apache Spark on Azure. With fully managed Spark clusters, it is used to process large workloads of data and also helps in data engineering, data exploring and also visualizing data using Machine learning.

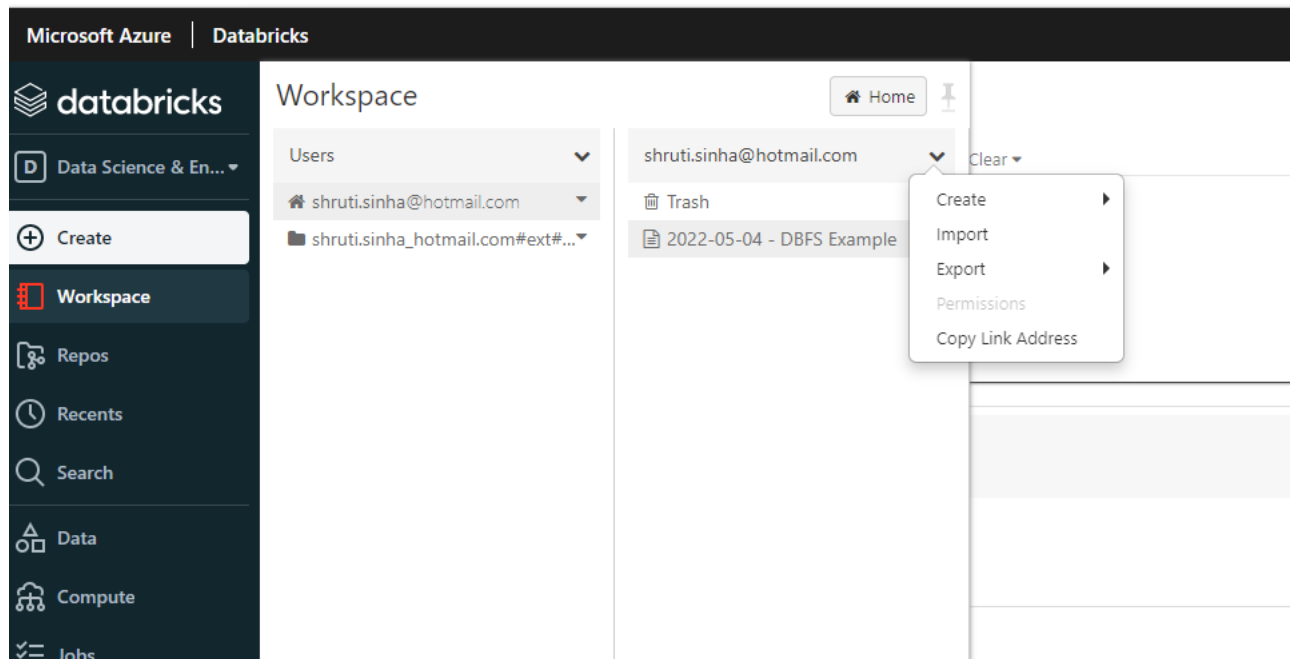
In this notebook, we will explore combining streaming and batch processing with a single pipeline. We will begin by defining the following logic:

- Ingest streaming JSON data from disk and write it to a Delta Lake Table **/activity/Bronze**
- perform a Stream-Static Join on the streamed data to add additional geographic data
- transform and load the data, saving it out to our Delta Lake Table **/activity/Silver**
- summarize the data through aggregation into the Delta Lake Table **/activity/Gold/groupedCounts**
- materialize views of our gold table through streaming plots and static queries

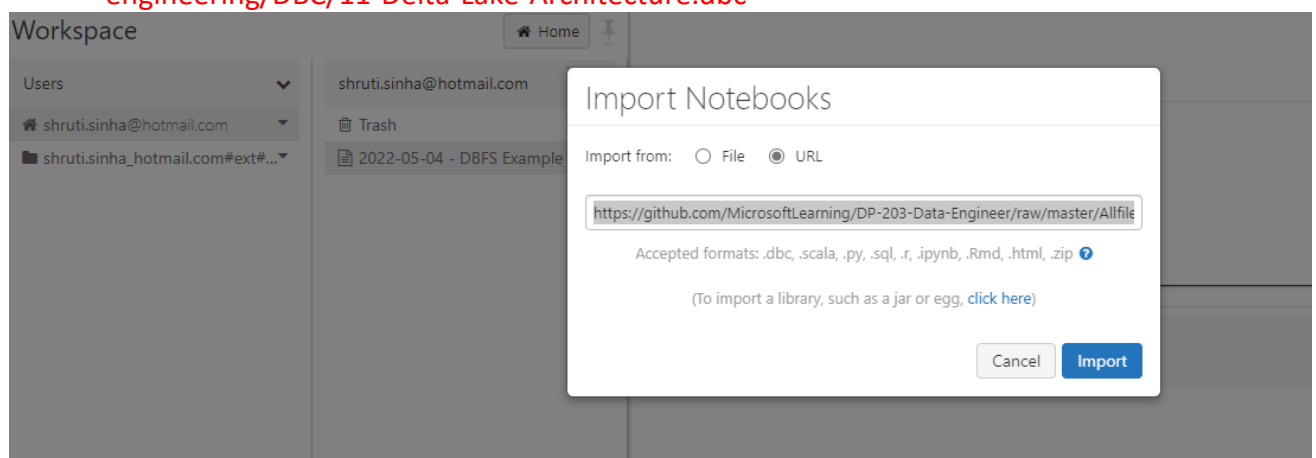
We will then demonstrate that by writing batches of data back to our bronze table, we can trigger the same logic on newly loaded data and propagate our changes automatically.

2. Exercise 1 – Ingest Data from GitHub

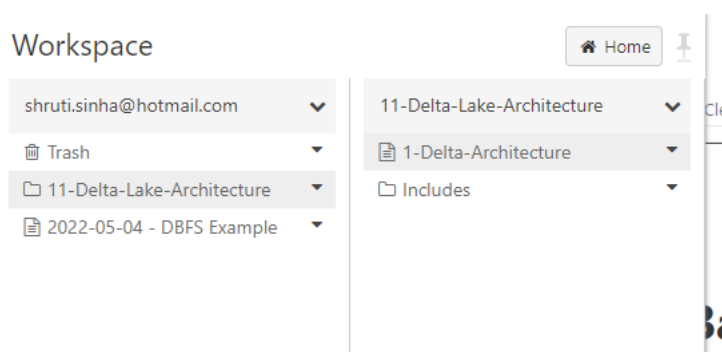
- In the Azure Databricks Workspace, in the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).
- In the pane that appears, select the arrow next to your name, and select **Import**.



- In the Import Notebooks dialog box, select the **URL** and paste in the following URL:
Paste Content
<https://github.com/MicrosoftLearning/DP-203-Data-Engineer/raw/master/Allfiles/microsoft-learning-paths-databricks-notebooks/data-engineering/DBC/11-Delta-Lake-Architecture.dbc>



- Click **Import**.
- Select the **11-Delta-Lake-Architecture** folder that appears.



3. Exercise: Run code in the 1-Delta-Architecture notebook

- Open the **1-Delta-Architecture notebook**.
- **Attach your cluster** to the notebook before following the instructions and running the cells it contains.

1-Delta-Architecture Python

bipeencus File Edit View: Standard Run All Clear

Cmd 1



Unifying Structured Streaming with Batch Jobs with Delta Lake

In this notebook, we will explore combining streaming and batch processing with a single pipeline. We will begin by defining the following logic:

- ingest streaming JSON data from disk and write it to a Delta Lake Table `/activity/Bronze`
- perform a Stream-Static Join on the streamed data to add additional geographic data
- transform and load the data, saving it out to our Delta Lake Table `/activity/Silver`
- summarize the data through aggregation into the Delta Lake Table `/activity/Gold/groupedCounts`
- materialize views of our gold table through streaming plots and static queries

We will then demonstrate that by writing batches of data back to our bronze table, we can trigger the same logic on newly loaded data and propagate our chan

Cmd 2

```
1 %run "../Includes/Classroom-Setup"
```

Cmd 3

Set up relevant Delta Lake paths

These paths will serve as the file locations for our Delta Lake tables.



Each streaming write has its own checkpoint directory.



You cannot write out new Delta files within a repository that contains Delta files. Note that our hierarchy here isolates each Delta table into its own directory

Cmd 4

```
1 activityPath = userhome + "/activity"
2
3 activityBronzePath = activityPath + "/Bronze"
4 activityBronzeCheckpoint = activityBronzePath + "/checkpoint"
```

- Initialized classroom variables & functions...

```
1 %run "../Includes/Classroom-Setup"
```

Command took 32.23 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:14:11 PM on bipeencus

Initialized classroom variables & functions...

Mounted datasets to `/mnt/training` from `wasbs://training@dbtraineastus.blob.core.windows.net/`

Created user-specific database

Using the database `shruti_sinha_hotmail_com_db`.

All done!

Cmd 3

- Set up relevant **Delta Lake paths**

```
Cmd 4
1 activityPath = userhome + "/activity"
2
3 activityBronzePath = activityPath + "/Bronze"
4 activityBronzeCheckpoint = activityBronzePath + "/checkpoint"
5
6 activitySilverPath = activityPath + "/Silver"
7 activitySilverCheckpoint = activitySilverPath + "/checkpoint"
8
9 activityGoldPath = activityPath + "/Gold"
10 groupedCountPath = activityGoldPath + "/groupedCount"
11 groupedCountCheckpoint = groupedCountPath + "/checkpoint"

Command took 0.02 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:15:16 PM on bipeenclus
```

- To reset the pipeline, run the following:

```
Cmd 6
1 dbutils.fs.rm(activityPath, True)

Out[7]: False

Command took 0.09 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:15:41 PM on bipeenclus
```

1. *Datasets Used*

This notebook will consume cell phone accelerometer data. Records have been downsampled so that the streaming data represents less than 3% of the total data being produced. The remainder will be processed as batches.

The following fields are present:

- Index
- Arrival_Time
- Creation_Time
- x
- y
- z
- User
- Model
- Device
- gt
- geolocation

2. *Define Schema*

For streaming jobs, we need to define our schema before we start.

```
Cmd 8

1 from pyspark.sql.types import StructField, StructType, LongType, StringType, DoubleType
2
3 schema = StructType([
4     StructField("Arrival_Time",LongType()),
5     StructField("Creation_Time",LongType()),
6     StructField("Device",StringType()),
7     StructField("Index",LongType()),
8     StructField("Model",StringType()),
9     StructField("User",StringType()),
10    StructField("geolocation",StructType([
11        StructField("city",StringType()),
12        StructField("country",StringType())
13    ])),
14    StructField("gt",StringType()),
15    StructField("id",LongType()),
16    StructField("x",DoubleType()),
17    StructField("y",DoubleType()),
18    StructField("z",DoubleType())
19 ])

Command took 0.03 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:16:15 PM on bipecnclus
```

3. *Define Streaming Load from Files in Blob*

Our streaming source directory has 36 JSON files of 5k records each saved in a repository. Here, we'll trigger processing on files one at a time.

```
Cmd 10

1 rawEventsDF = (spark
2     .readStream
3     .format("json")
4     .schema(schema)
5     .option("maxFilesPerTrigger", 1)
6     .load("/mnt/training/definitive-guide/data/activity-json/streaming"))

▶ rawEventsDF: pyspark.sql.dataframe.DataFrame = [Arrival_Time: long, Creation_Time: long ... 10 more fields]

Command took 0.58 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:16:45 PM on bipecnclus
```

4. *WRITE Stream using Delta Lake*

```
1 (rawEventsDF
2     .writeStream
3     .format("delta")
4     .option("checkpointLocation", activityBronzeCheckpoint)
5     .outputMode("append")
6     .start(activityBronzePath))

Cancel *** Running command...
🟢 Stream initializing...

Out[10]: <pyspark.sql.streaming.StreamingQuery at 0x7fc9d408a7f0>

Cmd 13
```

5. *Load Static Lookup Table*

Before enriching our bronze data, we will load a static lookup table for our country codes.

Here, we'll use a **parquet file** that contains **countries and their associated codes** and abbreviations.

While we can load this as a table (which will copy all files to the workspace and make it available to all users), here we'll manipulate it as a **DataFrame**.

```
Cmd 14
1 from pyspark.sql.functions import col
2
3 geoForLookupDF = (spark
4   .read
5   .format("parquet")
6   .load("/mnt/training/countries/ISOCountryCodes/ISOCountryLookup.parquet/")
7   .select(col("EnglishShortName").alias("country"), col("alpha3Code").alias("countryCode3"))

▶ (1) Spark Jobs
▶ geoForLookupDF: pyspark.sql.dataframe.DataFrame = [country: string, countryCode3: string]

Command took 0.50 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:17:58 PM on bipeenclus
```

6. Create QUERY tables (aka "silver tables")

Our current bronze table contains nested fields, as well as time data that has been encoded in non-standard unix time (Arrival_Time is encoded as milliseconds from epoch, while Creation_Time records nanoseconds between record creation and receipt).

We also wish to enrich our data with 3 letter country codes for mapping purposes, which we'll obtain from a join with our **geoForLookupDF**.

In order to parse the data in human-readable form, we create query/silver tables out of the raw data.

We will stream from our previous file write, define transformations, and rewrite our data to disk.

```
Cmd 14
1 from pyspark.sql.functions import col
2
3 geoForLookupDF = (spark
4   .read
5   .format("parquet")
6   .load("/mnt/training/countries/ISOCountryCodes/ISOCountryLookup.parquet/")
7   .select(col("EnglishShortName").alias("country"), col("alpha3Code").alias("countryCode3"))

▶ (1) Spark Jobs
▶ geoForLookupDF: pyspark.sql.dataframe.DataFrame = [country: string, countryCode3: string]

Command took 0.50 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:17:58 PM on bipeenclus
```

7. Create QUERY tables (aka "silver tables")

```
Cmd 19
1 (parsedEventsDF
2   .writeStream
3   .format("delta")
4   .option("checkpointLocation", activitySilverCheckpoint)
5   .outputMode("append")
6   .start(activitySilverPath))

Cancel
▶ (1) Spark Jobs
▶ Stream initializing...

Out[13]: <pyspark.sql.streaming.StreamingQuery at 0x7fc9d402e7f0>
```


Cmd 21

```
1 dbutils.fs.ls(activitySilverPath)
```

```
Out[14]: [FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/_delta_log/', name='_delta_log/', size=0, modificationTime=1651646946000),
FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/checkpoint/', name='checkpoint/', size=0, modificationTime=1651646922000),
FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/part-00000-3742dd2a-e2af-4d3a-8631-5889f77ebd82-c000.snappy.parquet', name='part-00000-3742dd
modificationTime=1651646939000),
FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/part-00000-a5a1e307-9a6c-41ff-a40a-8f1a2ef1028c-c000.snappy.parquet', name='part-00000-a5a1e3
modificationTime=1651646930000),
FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/part-00000-c314f6d5-9cd4-4e13-9d9d-6e2e3036fa49-c000.snappy.parquet', name='part-00000-c314f6
modificationTime=1651646933000),
FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/part-00000-dacb57f8-0006-462f-9a94-1b2eeabb8796-c000.snappy.parquet', name='part-00000-dacb57
modificationTime=1651646945000),
FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/part-00000-e4647221-8c16-4ac5-9dba-5555bda5d957-c000.snappy.parquet', name='part-00000-e46472
modificationTime=1651646936000),
FileInfo(path='dbfs:/user/shruti.sinha@hotmail.com/activity/Silver/part-00000-fd8393ae-825e-48b1-8dd0-c9bee4aa5d3b-c000.snappy.parquet', name='part-00000-fd8393
modificationTime=1651646942000)].
```

8. See list of active streams.

Cmd 23

```
1 for s in spark.streams.active:
2     print(s.id)
```

```
cd223faf-8786-485c-b5e2-cb015fcbc370
075ad3eb-b6a9-4563-b8f4-5855f695c298
```

Command took 0.03 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:19:31 PM on bipeenclus

Cmd 24

9. Gold Table: Grouped Count of Events

Here we read a stream of data from activitySilverPath and write another stream to activityGoldPath/groupedCount.

The data consists of a total counts of all event, grouped by hour, gt, and countryCode3.

Performing this aggregation allows us to reduce the total number of rows in our table from hundreds of thousands (or millions, once we've loaded our batch data) to dozens.

In cell cmd25 this can be seen as a materialized view of the streaming data.

Cmd 25

```
1 from pyspark.sql.functions import window, hour
2
3 (spark.readStream
4     .format("delta")
5     .load(activitySilverPath)
6     .groupBy(window("Arrival_Time", "60 minute"))
7     .count()
8     .withColumn("hour",hour(col("window.start")))
9     .drop("window")
10    .writeStream
11    .format("delta")
12    .option("checkpointLocation", groupedCountCh
13    .outputMode("complete")
14    .start(groupedCountPath))
```

Cancel

► (1) Spark Jobs

10. Gold Table: Grouped Count of Events

Here we read a stream of data from **activitySilverPath** and write another stream to **activityGoldPath/groupedCount**.

The data consists of a total counts of all event, grouped by **hour**, **gt**, and **countryCode3**.

Performing this aggregation allows us to reduce the total number of rows in our table from hundreds of thousands (or millions, once we've loaded our batch data) to dozens.

- In cell cmd25 this can be seen as a materialized view of the streaming data.

```
Cmd 27
1 spark.sql("""
2   DROP TABLE IF EXISTS grouped_count
3   """)
4 spark.sql("""
5   CREATE TABLE grouped_count
6   USING DELTA
7   LOCATION '{}'.format(groupedCountPath))
8   """)

Out[18]: DataFrame[]

Command took 0.46 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:20:34 PM on bipeencus
Cmd 28
```

- The gold Delta table we have just registered will perform a static read of the current state of the data each time we run the following query.

```
Cmd 29
1 %sql
2 SELECT * FROM grouped_count
```

▶ (2) Spark Jobs

Table Data Profile

	gt	countryCode3	count	hour
1	stairsdown	IND	440	12
2	stairsdown	DEU	2542	13
3	bike	NGA	3204	11
4	sit	NGA	2970	11
5	null	BRA	4110	14
6	null	IND	685	13
7	stairsdown	NGA	2810	11

Showing all 70 rows.

11. Materialized View: Windowed Count of Hourly gt Events

Cmd 31

```
1 (spark.readStream
2   .format("delta")
3   .load(activitySilverPath)
4   .createOrReplaceTempView("query_table")
5 )
```

Command took 0.06 seconds -- by shruti.sinha@hotmail.com at 5/4/2022, 12:21:34 PM on bipeencus

Cmd 32

```
1 %sql
2 SELECT gt, HOUR(Arrival_Time) hour, COUNT(*) total_events
3 FROM query_table
4 GROUP BY gt, HOUR(Arrival_Time)
5 ORDER BY hour
```

Cancel

▶ (1) Spark Jobs 

🟢 Stream initializing...

Cmd 33

Cmd 32

```
1 %sql
2 SELECT gt, HOUR(Arrival_Time) hour, COUNT(*) total_events
3 FROM query_table
4 GROUP BY gt, HOUR(Arrival_Time)
5 ORDER BY hour
```

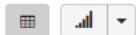
Cancel **

▶ (1) Spark Jobs 

▶ 🟢 display_query_1 (id: fa861148-b696-468c-9c0e-4909ff247aee) Last updated: 10 seconds ago

	gt	hour	total_events
1	null	10	4420
2	bike	10	2833
3	sit	10	3077
4	walk	10	3423
5	stand	10	3010
6	stairsdown	10	2545
7	stairsup	10	2255

Showing all 35 rows.




Cmd 33

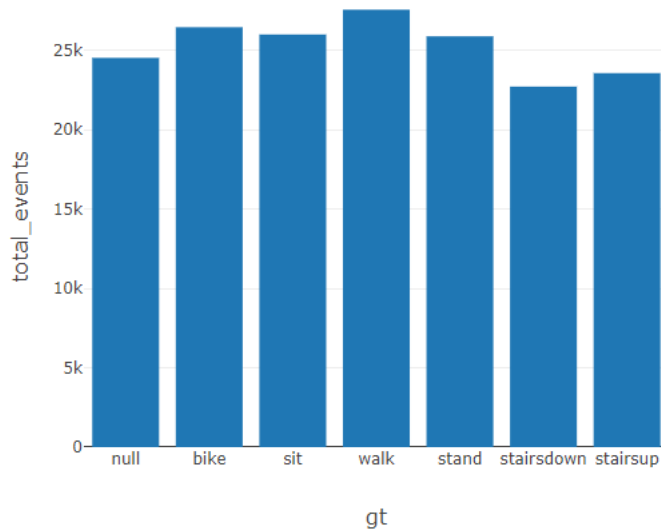
Cmd 32

```
1 %sql
2 SELECT gt, HOUR(Arrival_Time) hour, COUNT(*) total_events
3 FROM query_table
4 GROUP BY gt, HOUR(Arrival_Time)
5 ORDER BY hour
```

Cancel

▶ (1) Spark Jobs

▶  display_query_1 (id: fa861148-b696-468c-9c0e-4909ff247aee) *Last updated: 10 seconds ago*



Plot Options...

Cmd 33