

**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](#_Toc102632905)

[2. Exercise 1 – Ingest Data from GitHub 3](#_Toc102632906)

[3. Exercise: Run code in the 1-Delta-Architecture notebook 5](#_Toc102632907)

[1. Datasets Used 6](#_Toc102632908)

[2. Define Schema 6](#_Toc102632909)

[3. Define Streaming Load from Files in Blob 7](#_Toc102632910)

[4. WRITE Stream using Delta Lake 7](#_Toc102632911)

[5. Load Static Lookup Table 7](#_Toc102632912)

[6. Create QUERY tables (aka "silver tables") 8](#_Toc102632913)

[7. Create QUERY tables (aka "silver tables") 8](#_Toc102632914)

[8. See list of active streams. 9](#_Toc102632915)

[9. Gold Table: Grouped Count of Events 9](#_Toc102632916)

[10. Gold Table: Grouped Count of Events 10](#_Toc102632917)

[11. Materialized View: Windowed Count of Hourly gt Events 10](#_Toc102632918)

# 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](https://spark.apache.org/) 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](https://spark.apache.org/docs/0.9.1/scala-programming-guide.html), 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.

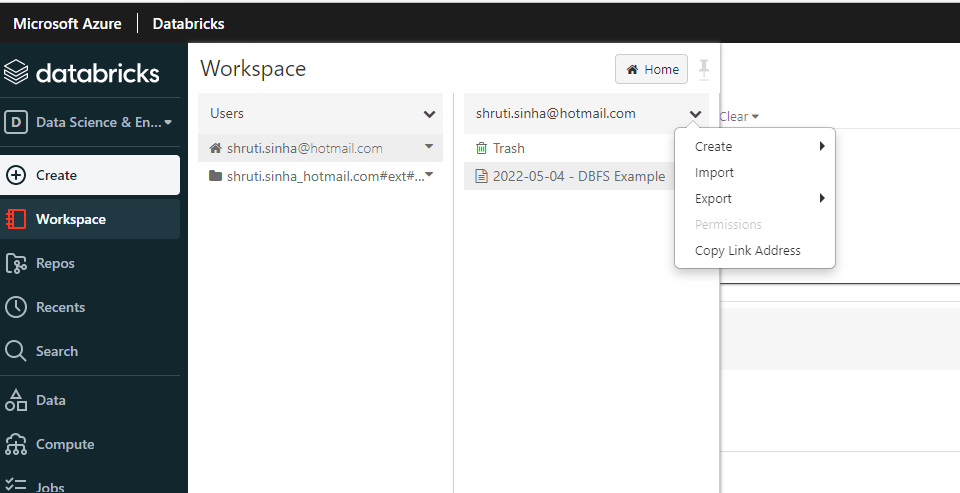
In this notebook, we will explore combining streaming and batch processing with a single pipeline. We will begin by defining the following logic:

* 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.

# Exercise 1 – Ingest Data from GitHub

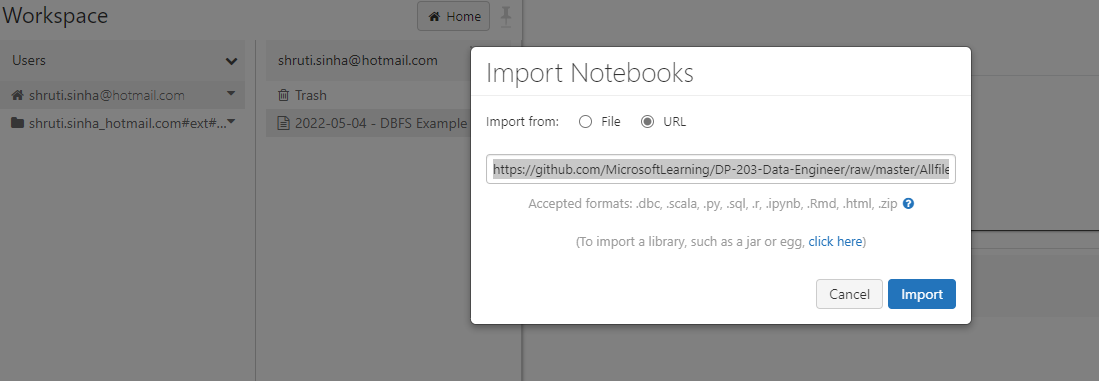
* In the Azure Databricks Workspace, in the left pane, select **Workspace >** Users, and select your username (the entry with the house icon).
* In the pane that appears, select the arrow next to your name, and select **Import.**



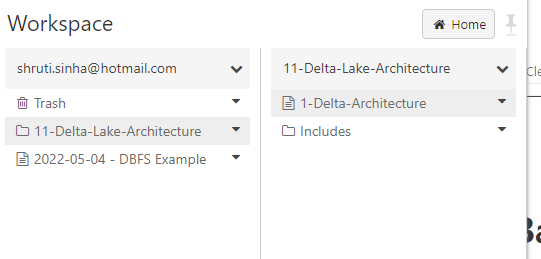
* In the Import Notebooks dialog box, select the **URL** and paste in the following URL:

Paste Content

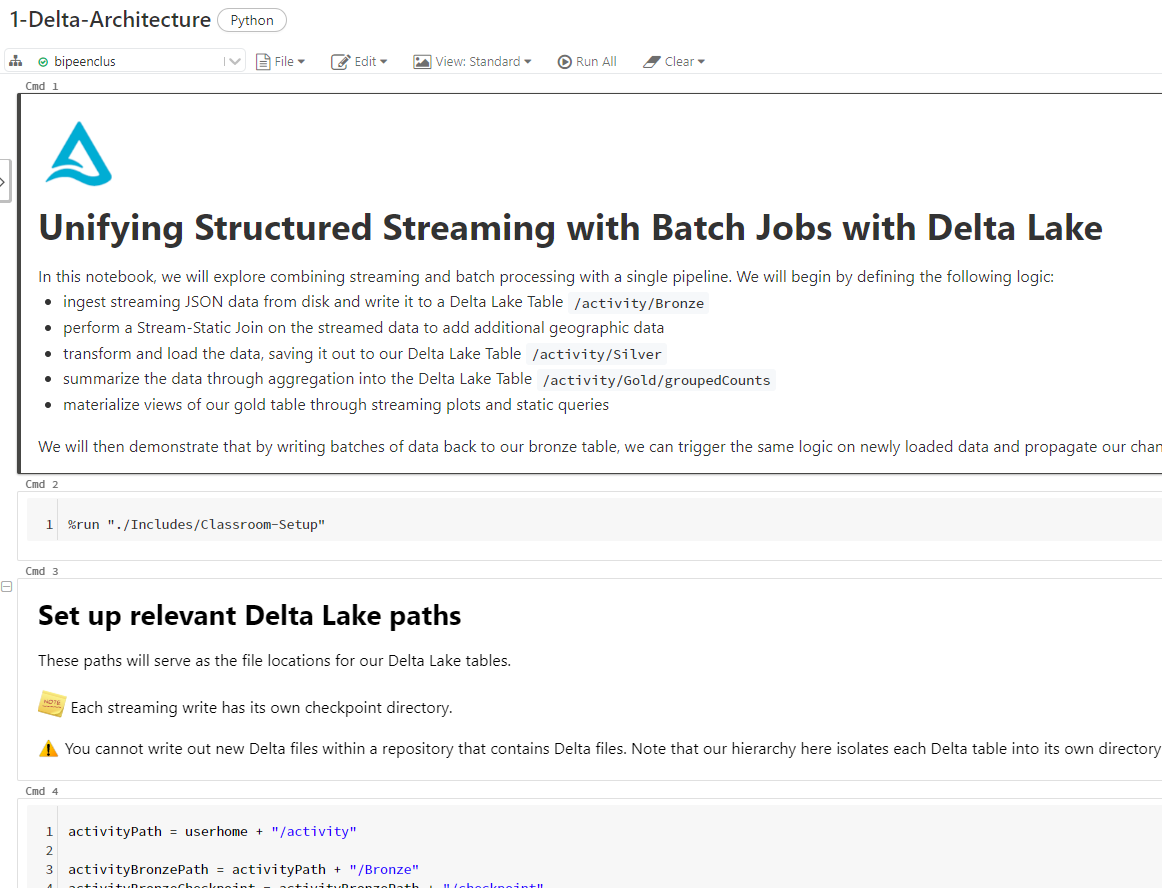
https://github.com/MicrosoftLearning/DP-203-Data-Engineer/raw/master/Allfiles/microsoft-learning-paths-databricks-notebooks/data-engineering/DBC/11-Delta-Lake-Architecture.dbc

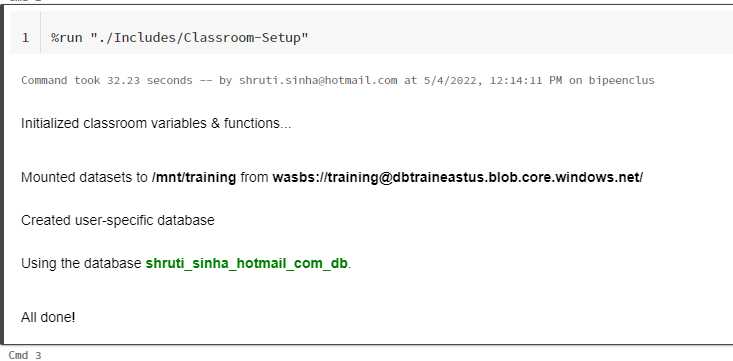


* Click **Import.**
* Select the **11-Delta-Lake-Architecture** folder that appears.

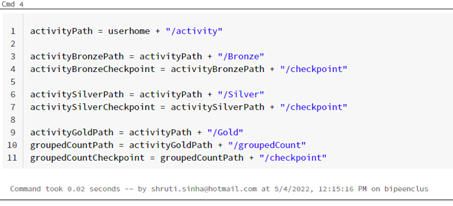


# Exercise: Run code in the 1-Delta-Architecture notebook

* Open the **1-Delta-Architecture noteboo**k.
* **Attach your cl**uster to the notebook before following the instructions and running the cells it contains. 
* Initialized classroom variables & functions...



* Set up relevant **Delta Lake paths**



* To reset the pipeline, run the following:



## 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

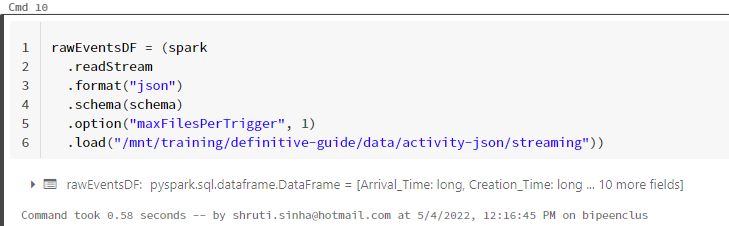
## Define Schema

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



## 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.



## WRITE Stream using Delta Lake



## 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 c**odes 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**.



## 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.



## Create QUERY tables (aka "silver tables")





## See list of active streams.



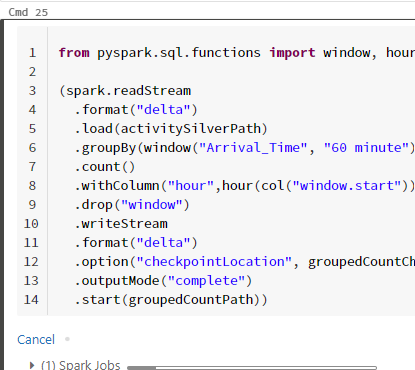
## Gold Table: Grouped Count of Events

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.



## Gold Table: Grouped Count of Events

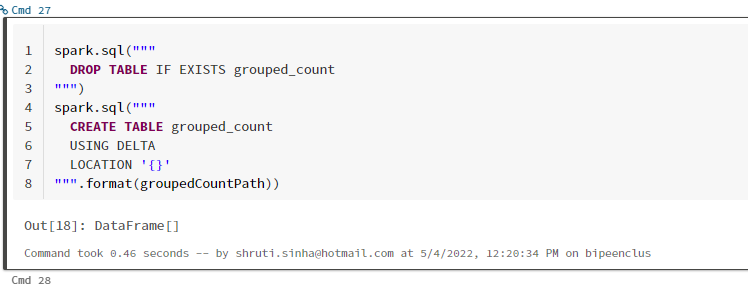
Here we read a stream of data from **activitySilverPath** and write another stream to **activityGoldPath/groupedCount**.

The data consists of a total counts of all event, grouped by **hour, gt**, and **countryCode**3.

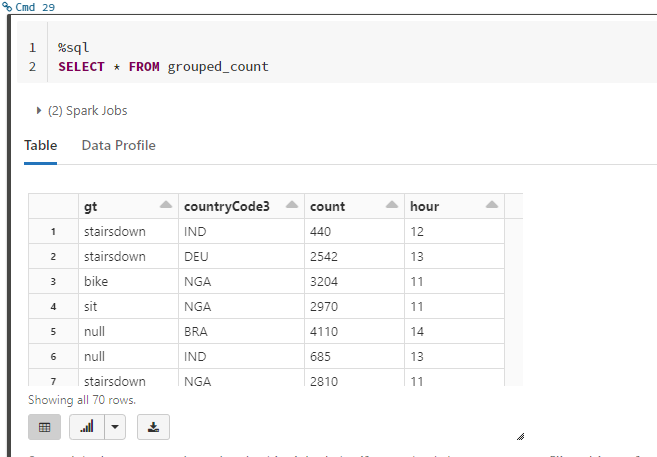
Performing this aggregation allows us to reduce the total number of rows in our table from hundreds of thousands (or millions, once we've loaded our batch data) to dozens.

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

\



* 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.



## Materialized View: Windowed Count of Hourly gt Events



