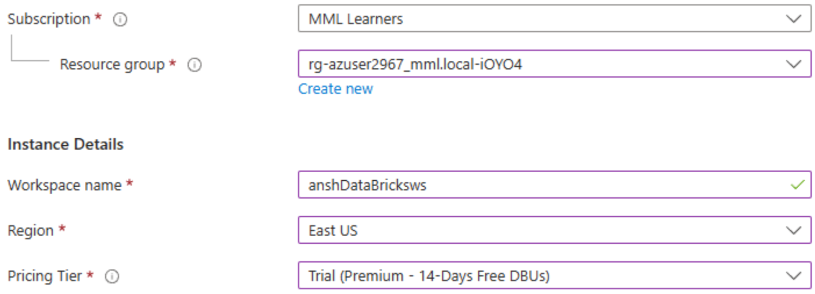
Ansh Ranjan
Azure Databricks – All Exercises

Exercise 1 – Settings up DataBricks and Spark Basics

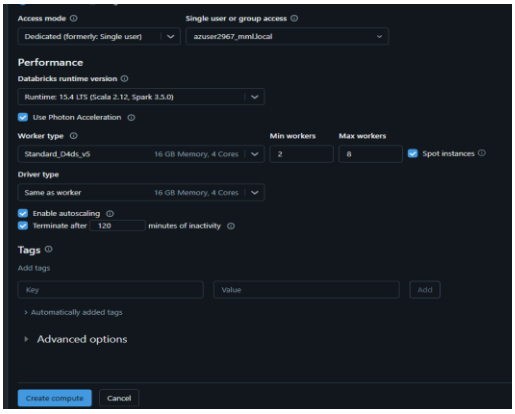
TASK 1: Create a new Azure DataBricks workspace

1. Go to Azure Portal > Azure DataBricks > Create > Enter details > Review and Create

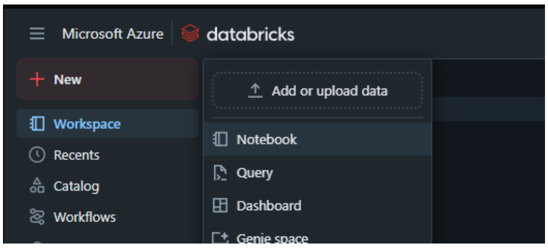


TASK 2: Launch a spark cluster and explore databricks interface

1. Launch your Databricks workspace > Computer side tab > Create Compute



2. Now click on New > Notebook and you will have your notebook ready in your workspace



The **Azure Databricks workspace** provides a unified environment for data engineering, data science, and machine learning. The main parts of the interface include:

1. Workspace

- Organize notebooks, libraries, and workflows.
- Create folders and share them with users or groups.

2. Notebooks

- Interactive notebooks supporting **Python**, **SQL**, **Scala**, and **R**.
- Run code in cells and visualize data easily.

3. Clusters

- Spin up Spark clusters for running jobs or interactive analysis.

- Choose autoscaling and runtime version (with Delta, ML, or GPU support).

4. Jobs

- Schedule notebooks or workflows.
- Automate ETL, ML training, or batch jobs.

5. Data

- Browse databases, tables, and files.
- Supports Unity Catalogue (if enabled) for secure, centralized governance.

6. Repos (Git Integration)

- Connect to GitHub or Azure DevOps to version-control notebooks and code.

TASK 3: Run Basic Spark commands

1. Running spark.version command in first cell

```
spark.version
'3.5.0'
```

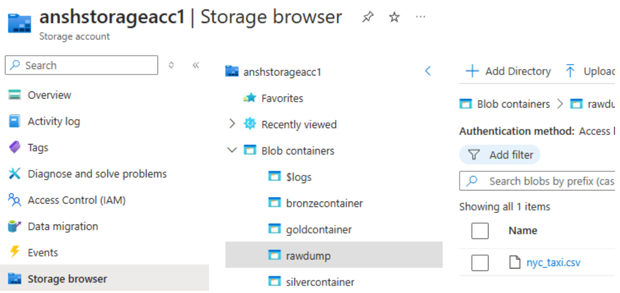
```
spark.conf.get("spark.master")
'spark://10.139.64.4:7077'
```

2. Now that we know our databricks workspace is ready we can start performing ETL tasks.

Exercise 2 – Data Ingestion

TASK 1: Load a dataset into your Databricks using Spark

1. Our data resides in a 'rawdump' named container in our storage account.



2. Creating a spark session

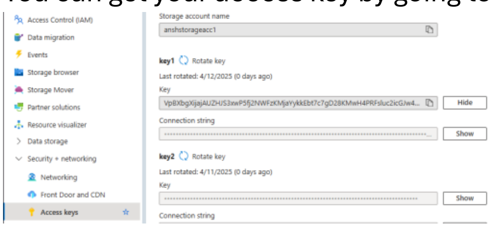
```
from pyspark.sql import SparkSession

spark=SparkSession.builder.appName('nyc_taxi').getOrCreate()
```

3. Now we have to mount our data point to our workspace.

```
dbutils.fs.mount(
    source="wasbs://rawdump@anshstorageacc1.blob.core.windows.net/",
    mount_point="/mnt/rawdata",
    extra_configs={f"fs.azure.account.key.anshstorageacc1.blob.core.windows.net":"VpBXbgXijajAUZHJS3xwP5fj2NWFzKMjaYykkEbt7c7gD28KMwH4PRFsluc2icGJw49Lb6x6KVMD+AStrARmBA=="}
)
True
```

You can get your access key by going to storage account > Security + networking > Access Keys



4. Reading data into a dataframe

```
Just now (1s)
%fs ls dbfs:/mnt/rawdata
```

Table

	path	name	size	modificationTime
1	dbfs:/mnt/rawdata/nyc_taxi.csv	nyc_taxi.csv	200589097	1744448453000

```
df_bronze = spark.read.csv("dbfs:/mnt/rawdata/nyc_taxi.csv", header=True, inferSchema=True)
```

(2) Spark Jobs

TASK 2: Inspect the data

1. Displaying the data

```
df_bronze.show()
```

(1) Spark Jobs

```
+---------+---------+-------------------+-------------------+---------------+------------------+------------------+------------------+------------------+------------------+-----------
-+
|       id|vendor_id|    pickup_datetime|   dropoff_datetime|passenger_count|  pickup_longitude|   pickup_latitude| dropoff_longitude|  dropoff_latitude|store_and_fwd_flag|trip_duratio
n|
+---------+---------+-------------------+-------------------+---------------+------------------+------------------+------------------+------------------+------------------+-----------
-+
|id2875421|        2|2016-03-14 17:24:55|2016-03-14 17:32:30|              1| -73.9821548461914| 40.76793670654297|-73.96463012695312|40.765602111816406|                 N|         45
5|
|id2377394|        1|2016-06-12 00:43:35|2016-06-12 00:54:38|              1|-73.98041534423828|40.738563537597656|-73.99948120117188| 40.73115158081055|                 N|         66
3|
|id3858529|        2|2016-01-19 11:35:24|2016-01-19 12:10:48|              1| -73.9790267944336|40.763938903808594|-74.00533294677734|40.710086822509766|                 N|        212
```

2. Dataframe schema

```
df_bronze.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- vendor_id: integer (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- trip_duration: integer (nullable = true)
```

3. Number of rows

```
df_bronze.count()
```

(2) Spark Jobs

```
1458644
```

TASK 3: Perform data cleaning

1. Dropping null values from the df

```
df_bronze.dropna()
```

```
DataFrame[id: string, vendor_id: int, pi
double, dropoff_latitude: double, store_
```

Now there should not be any null values

```
from pyspark.sql.functions import col, sum, when

null_counts = df_bronze.select([
    sum(when(col(c).isNull(), 1).otherwise(0)).alias(c)
    for c in df_bronze.columns
])

null_counts.show()
```

(2) Spark Jobs

null_counts: pyspark.sql.dataframe.DataFrame = [id: long, vendor_id: long ... 9 more fields]

```
+---+---------+---------------+----------------+---------------+----------------+---------------+-----------------+----------------+------------------+-------------+
| id|vendor_id|pickup_datetime|dropoff_datetime|passenger_count|pickup_longitude|pickup_latitude|dropoff_longitude|dropoff_latitude|store_and_fwd_flag|trip_duration|
+---+---------+---------------+----------------+---------------+----------------+---------------+-----------------+----------------+------------------+-------------+
|  0|        0|              0|               0|              0|               0|              0|                0|               0|                 0|            0|
+---+---------+---------------+----------------+---------------+----------------+---------------+-----------------+----------------+------------------+-------------+
```

2. As we saw earlier, our pickup and dropoff columns are of double data type. We need to convert them into standard datetime format

```
from pyspark.sql.functions import to_timestamp

df_bronze_converted = df_bronze.withColumn("pickup_datetime", to_timestamp("pickup_datetime", "yyyy-MM-dd HH:mm:ss")) \
                                .withColumn("dropoff_datetime", to_timestamp("dropoff_datetime", "yyyy-MM-dd HH:mm:ss"))

df_bronze_converted.printSchema()
```

```
df_bronze_converted: pyspark.sql.dataframe.DataFrame = [id: string, vendor_id: integer ... 9 more fields]
root
 |-- id: string (nullable = true)
 |-- vendor_id: integer (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- trip_duration: integer (nullable = true)
```

3. Checking for any rows where dropoff time might be before pickup time for data quality

```
count = df_bronze_converted.filter(col("pickup_datetime") > col("dropoff_datetime")).count()
print(f"The number of rows with pickup_datetime > dropoff_datetime is: {count}")
```

```
(2) Spark Jobs
The number of rows with pickup_datetime > dropoff_datetime is: 0
```

EXERCISE 3: Data Transformation

TASK 1 and 2: Apply transformation to data (filtering, grouping, etc)

1. Creating a new column trip_duration as difference of pickup and dropoff time in minutes

```
df_bronze_with_duration = df_bronze_converted.withColumn(
    "trip_duration_minutes",
    round((unix_timestamp("dropoff_datetime") - unix_timestamp("pickup_datetime")) / 60, 2)
)
```

```
df_bronze_with_duration.select("pickup_datetime", "dropoff_datetime", "trip_duration_minutes").show(5)
```

```
(1) Spark Jobs
+-------------------+-------------------+---------------------+
|    pickup_datetime|   dropoff_datetime|trip_duration_minutes|
+-------------------+-------------------+---------------------+
|2016-03-14 17:24:55|2016-03-14 17:32:30|                 7.58|
|2016-06-12 00:43:35|2016-06-12 00:54:38|                11.05|
|2016-01-19 11:35:24|2016-01-19 12:10:48|                 35.4|
|2016-04-06 19:32:31|2016-04-06 19:39:40|                 7.15|
|2016-03-26 13:30:55|2016-03-26 13:38:10|                 7.25|
+-------------------+-------------------+---------------------+
only showing top 5 rows
```

2. Creating new columns day_od_week and Hour_of_day

```
from pyspark.sql.functions import dayofweek, hour

df_bronze_with_duration = df_bronze_with_duration\
    .withColumn("day_of_week", dayofweek(col("pickup_datetime")))\
    .withColumn("hour_of_day", hour(col("pickup_datetime")))

df_bronze_with_duration = df_bronze_with_duration\
    .withColumn("day_of_week", dayofweek(col("pickup_datetime")))\
    .withColumn("hour_of_day", hour(col("pickup_datetime")))
```

3. Finding number of trips per number of passengers

```
df_passenger_grouped = df_bronze_with_duration.groupBy("passenger_count") \
    .agg(count("*").alias("record_count")) \
    .orderBy("passenger_count")

df_passenger_grouped.show()
```

```
(2) Spark Jobs
df_passenger_grouped: pyspark.sql.dataframe.DataFrame = [passenger_count: integer, record_count: l
+---------------+------------+
|passenger_count|record_count|
+---------------+------------+
|              0|          60|
|              1|     1033540|
|              2|      210318|
|              3|       59896|
|              4|       28404|
|              5|       78088|
|              6|       48333|
|              7|           3|
|              8|           1|
|              9|           1|
+---------------+------------+
```

4. Grouping data by day of week to get number of trips made for each day of the week

```
df_week_grouped = df_bronze_with_duration.groupBy("day_of_week") \
    .agg(count("*").alias("Number_of_trips")) \
    .orderBy("day_of_week")

df_week_grouped.show()
```

```
▸ (2) Spark Jobs
▸ df_week_grouped: pyspark.sql.dataframe.DataFrame = [day_of_week: integer, Number_of_trips: long]

+-----------+---------------+
|day_of_week|Number_of_trips|
+-----------+---------------+
|          1|         195366|
|          2|         187418|
|          3|         202749|
|          4|         210136|
|          5|         218574|
|          6|         223533|
|          7|         220868|
+-----------+---------------+
```

EXERCISE 4 – Data Storage and Retrieval

TASK 1: save the transformed data into Azure blob

1. Mounting gold layer storage container

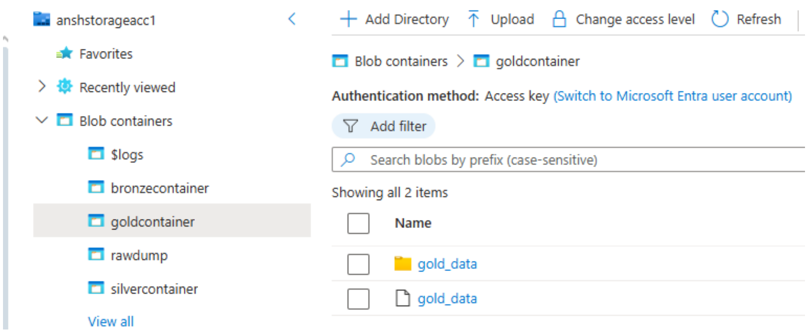
```
dbutils.fs.mount(
    source="wasbs://goldcontainer@anshstorageacc1.blob.core.windows.net/",
    mount_point="/mnt/goldlayer",
    extra_configs={f"fs.azure.account.key.anshstorageacc1.blob.core.windows.net":"VpBXbgXijajAUZHJS3xwP5fj2NWFzKMjaYykkEbt7c7gD28KMwH4PRFsluc2icGJw49Lb6x6KVMD+AStrARmBA=="}
)

True
```

2. Writing the dataframe in form of Delta Tables in Blob container

```
df_bronze_with_duration.write.format("delta").mode("overwrite").save("dbfs:/mnt/goldlayer/gold_data")
```

3. Checking data in gold container



TASK 2: Read the saved data back into df

1. Reading the delta table back as a new df

```
read_df = spark.read.format("delta").load("dbfs:/mnt/goldlayer/gold_data")
read_df.show(5)
```

```
▸ (1) Spark Jobs
▸ read_df: pyspark.sql.dataframe.DataFrame = [id: string, vendor_id: integer ... 12 more fields]

+---------+---------+-------------------+-------------------+---------------+------------------+-----------------+------------------+-----------------+------------------+-------------
-------------------+-----------+-----------+
|       id|vendor_id|    pickup_datetime|   dropoff_datetime|passenger_count|  pickup_longitude|  pickup_latitude| dropoff_longitude| dropoff_latitude|store_and_fwd_flag|trip_duration|t
rip_duration_minutes|day_of_week|hour_of_day|
+---------+---------+-------------------+-------------------+---------------+------------------+-----------------+------------------+-----------------+------------------+-------------+-
-------------------+-----------+-----------+
|id3380832|        1|2016-04-09 09:56:27|2016-04-09 10:11:12|              1|-73.97080993652344| 40.75213623046875|-73.99332427978516| 40.7461051940918|                 N|          885|
14.75|          7|          9|
|id0033606|        1|2016-04-22 13:59:39|2016-04-22 14:02:47|              1| -73.9861831665039| 40.76225662231445|-73.97857666015625|40.759071350097656|                 N|          188|
```

TASK 3: Explore Databricks Storage Options

Azure Databricks Storage Options

1. **DBFS (Databricks File System)**
   - Managed storage layer built into Databricks.
   - Mounts your cloud storage as /dbfs/.

- Easy for quick data loading and temp storage.

```
%fs ls
```

Table

	path	name	size	modificationTime
1	dbfs:/Volume/	Volume/	0	0
2	dbfs:/Volumes/	Volumes/	0	0
3	dbfs:/databricks-dataset...	databricks-dataset...	0	0
4	dbfs:/databricks-results/	databricks-results/	0	0
5	dbfs:/mnt/	mnt/	0	1744452793000
6	dbfs:/volume/	volume/	0	0
7	dbfs:/volumes/	volumes/	0	0

2. **Mounting Azure Storage (Blob or ADLS)**
    - Mount external storage (Blob or ADLS Gen2) to /mnt/your-mount-name.
    - Allows persistent storage with access to raw, bronze, silver, gold layers.
    - Uses dbutils.fs.mount() with access keys or service principals.

```
dbutils.fs.mount(
    source="wasbs://goldcontainer@anshstorageacc1.blob.core.windows.net/",
    mount_point="/mnt/goldlayer",
    extra_configs={f"fs.azure.account.key.anshstorageacc1.blob.core.windows
)
True
```

3. **Direct Access with ABFS or WASBS URLs**
    - No mount required.
    - Example: "abfss://container@storage.dfs.core.windows.net/"
    - Best for secure, scalable access with Unity Catalog.
4. **External Tables in Data Lake (Delta)**
    - Store Delta tables in ADLS or Blob and register them in Hive or Unity Catalog.
    - Enables scalable data lake architecture (bronze/silver/gold).

EXERCISE 5 – Advanced Spark Topics and Optimization

TASK 1: Implement caching and Broadcasting

1. Caching a dataframe in memory allows for faster reuse
2. To cache a dataframe run df.cache()

```
df_bronze_with_duration.cache()
df_bronze_with_duration.count()
```

3. If we are joining a large dataframe with a smaller one. We can broadcast the smaller dataframe across all worker nodes to avoid shuffling.

```
from pyspark.sql.functions import broadcast

df_joined = df_bronze_with_duration.join(
    broadcast(df_small_lookup), on="some_key", how="left"
)
```

TASK 2: Use spark's advanced concepts

1. Use DataFrames and Spark SQL (not RDDs)

- DataFrames are optimized by **Catalyst** and **Tungsten**, making them much faster.
- Avoid low-level RDDs unless absolutely necessary.

2. Cache / Persist Smartly

- Use .cache() or .persist() **only** when you reuse a DataFrame **multiple times**.
- Don't over-cache — memory is limited!

3. Broadcast Small Lookup Tables

- When joining with small datasets (< 100 MB), use:

```
from pyspark.sql.functions import broadcast
df_large.join(broadcast(df_small), "key")
```

4. Filter Early, Select Only Needed Columns

- Apply .filter() and .select() early to reduce data volume.

```
df.select("col1", "col2").filter("col1 > 100")
```

5. Partition Smartly

- Use .repartition() to distribute load for wide transformations or large joins.
- Use .coalesce(1) **only** for final small outputs (e.g., exports).

```
df = df.repartition(10, "some_column")
```

6. Use Delta Lake Format

- Delta tables are faster for reads, support ACID, schema evolution, and time travel.
- Use OPTIMIZE and ZORDER to improve read performance.

```
OPTIMIZE my_table ZORDER BY (col_name)
```

7. Avoid Exploding Joins & Shuffles

- Shuffles are expensive (joins, groupBy, distinct).
- Repartition wisely to avoid skew.

8. Monitor with Spark UI

- Always check stages and tasks in **Spark UI** (or the **Databricks Job Run view**).
- Look out for:
  - Long tasks
  - Skewed stages
  - Too many small files