Antra

People · Process · Performance

Agenda

- Working with Databricks using Python/PySpark

Working with Databricks using SparkSession

Antra

SparkSession

- The entry point to programming in Spark with the Dataset and DataFrame API.
- Can be used to
  - create DataFrame
  - register DataFrame as tables
  - execute SQL over tables
  - cache tables
  - read parquet files
- We can build a SparkSession using the Builder API

```
>>> spark = SparkSession.builder \
...     .master("local") \
...     .appName("Word Count") \
...     .config("spark.some.config.option", "some-value") \
...     .getOrCreate()
```

SparkSession Methods

createDataFrame(data[, schema, …])	Creates a **DataFrame** from an **RDD**, a list or a **pandas.DataFrame**.
getActiveSession()	Returns the active SparkSession for the current thread, returned by the builder
newSession()	Returns a new SparkSession as new session, that has separate SQLConf, registered temporary views and UDFs, but shared SparkContext and table cache.
range(start[, end, step, numPartitions])	Create a **DataFrame** with single **pyspark.sql.types.LongType** column named id, containing elements in a range from start to end (exclusive) with step value step.
sql(sqlQuery)	Returns a **DataFrame** representing the result of the given query.
stop()	Stop the underlying **SparkContext**.
table(tableName)	Returns the specified table as a **DataFrame**.

SparkSession Attributes

builder	A class attribute having a **Builder** to construct **SparkSession** instances.
catalog	Interface through which the user may create, drop, alter or query underlying databases, tables, functions, etc.
conf	Runtime configuration interface for Spark.
read	Returns a **DataFrameReader** that can be used to read data in as a **DataFrame**.
readStream	Returns a **DataStreamReader** that can be used to read data streams as a streaming **DataFrame**.
sparkContext	Returns the underlying **SparkContext**.
streams	Returns a **StreamingQueryManager** that allows managing all the **StreamingQuery** instances active on this context.
udf	Returns a **UDFRegistration** for UDF registration.
version	The version of Spark on which this application is running.

Read Data in Azure Databricks

csv: dbfs:/databricks-datasets/COVID/covid-19-data/us-counties.csv
json: dbfs:/databricks-datasets/learning-spark-v2/blogs.json
tsv:dbfs:/databricks-datasets/wikipedia-datasets/data-001/pageviews/raw/pageviews_by_second.tsv

Antra

CSV Example

- Displaying all the files under a path in DBFS

```
1  %fs ls databricks-datasets/COVID/covid-19-data/
```

	path	name	size
1	dbfs:/databricks-datasets/COVID/covid-19-data/.git/	.git/	0
2	dbfs:/databricks-datasets/COVID/covid-19-data/.github/	.github/	0
3	dbfs:/databricks-datasets/COVID/covid-19-data/.gitignore	.gitignore	10
4	dbfs:/databricks-datasets/COVID/covid-19-data/LICENSE	LICENSE	1289
5	dbfs:/databricks-datasets/COVID/covid-19-data/NEW-YORK-DEATHS-METHODOLOGY.md	NEW-YORK-DEATHS-METHODOLOGY.md	2771
6	dbfs:/databricks-datasets/COVID/covid-19-data/NYT-readme.md	NYT-readme.md	1748
7	dbfs:/databricks-datasets/COVID/covid-19-data/PROBABLE-CASES-NOTE.md	PROBABLE-CASES-NOTE.md	3162
8	dbfs:/databricks-datasets/COVID/covid-19-data/README.md	README.md	22959
9	dbfs:/databricks-datasets/COVID/covid-19-data/colleges/	colleges/	0
10	dbfs:/databricks-datasets/COVID/covid-19-data/excess-deaths/	excess-deaths/	0
11	dbfs:/databricks-datasets/COVID/covid-19-data/live/	live/	0
12	dbfs:/databricks-datasets/COVID/covid-19-data/mask-use/	mask-use/	0
13	dbfs:/databricks-datasets/COVID/covid-19-data/us-counties-recent.csv	us-counties-recent.csv	3977154
14	dbfs:/databricks-datasets/COVID/covid-19-data/us-counties.csv	us-counties.csv	45452100
15	dbfs:/databricks-datasets/COVID/covid-19-data/us-states.csv	us-states.csv	703102
16	dbfs:/databricks-datasets/COVID/covid-19-data/us.csv	us.csv	10269

Showing all 16 rows.

DataFrameReader

- We can use DataFrameReader to read many different types of data and turn them into DataFrames.

DataFrameReader.csv(path[, schema, sep, …])	Loads a CSV file and returns the result as a **DataFrame**.
DataFrameReader.format(source)	Specifies the input data source format.
DataFrameReader.jdbc(url, table[, column, …])	Construct a **DataFrame** representing the database table named table accessible via JDBC URL url and connection properties.
DataFrameReader.json(path[, schema, …])	Loads JSON files and returns the results as a **DataFrame**.
DataFrameReader.load([path, format, schema])	Loads data from a data source and returns it as a **DataFrame**.
DataFrameReader.option(key, value)	Adds an input option for the underlying data source.
DataFrameReader.options(**options)	Adds input options for the underlying data source.
DataFrameReader.orc(path[, mergeSchema, …])	Loads ORC files, returning the result as a **DataFrame**.
DataFrameReader.parquet(*paths, **options)	Loads Parquet files, returning the result as a **DataFrame**.
DataFrameReader.schema(schema)	Specifies the input schema.
DataFrameReader.table(tableName)	Returns the specified table as a **DataFrame**.

CSV Example

- Use **%fs head** to peek at the beginning of the file

Cmd 21

```
1  %fs head databricks-datasets/COVID/covid-19-data/us-counties.csv
```

```
[Truncated to first 65536 bytes]
date,county,state,fips,cases,deaths
2020-01-21,Snohomish,Washington,53061,1,0
2020-01-22,Snohomish,Washington,53061,1,0
2020-01-23,Snohomish,Washington,53061,1,0
2020-01-24,Cook,Illinois,17031,1,0
2020-01-24,Snohomish,Washington,53061,1,0
2020-01-25,Orange,California,06059,1,0
2020-01-25,Cook,Illinois,17031,1,0
2020-01-25,Snohomish,Washington,53061,1,0
2020-01-26,Maricopa,Arizona,04013,1,0
2020-01-26,Los Angeles,California,06037,1,0
2020-01-26,Orange,California,06059,1,0
2020-01-26,Cook,Illinois,17031,1,0
2020-01-26,Snohomish,Washington,53061,1,0
2020-01-27,Maricopa,Arizona,04013,1,0
2020-01-27,Los Angeles,California,06037,1,0
2020-01-27,Orange,California,06059,1,0
2020-01-27,Cook,Illinois,17031,1,0
2020-01-27,Snohomish,Washington,53061,1,0
2020-01-28,Maricopa,Arizona,04013,1,0
```

CSV Example

- Using **read.csv()** we get a dataframe with 6 columns

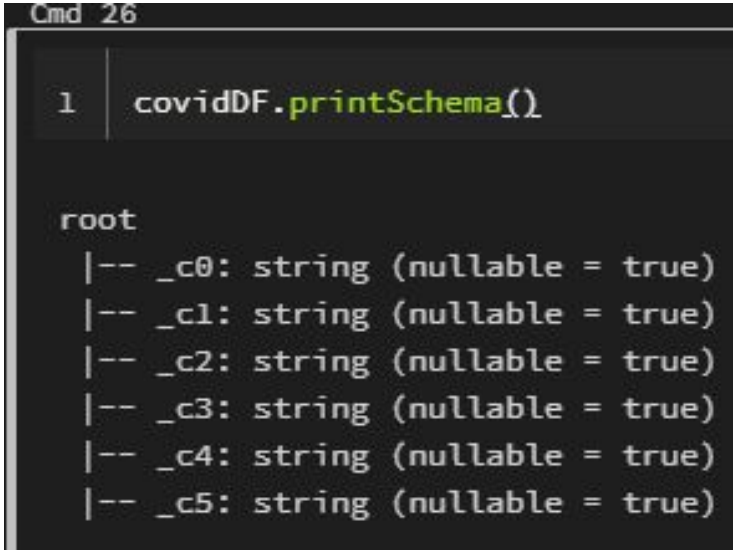
```
Cmd 25

1  covidDF = (spark.read
2              .csv("/databricks-datasets/COVID/covid-19-data/us-counties.csv")
3            )

▸ (1) Spark Jobs
▾ covidDF: pyspark.sql.dataframe.DataFrame
      _c0: string
      _c1: string
      _c2: string
      _c3: string
      _c4: string
      _c5: string
```

CSV Example

- Dataframe has many methods and attributes
- **printSchema** will print the schema of the DF in tree format



Antra

CSV Example

- Recall when we peeked the data, the first row is the header
- We can tell spark to treat first row as header by setting “header” to “true” in option

Cmd 34

```
1  covidDF = (spark.read
2             .option("header","true")
3             .csv("/databricks-datasets/COVID/covid-19-data/us-counties.csv")
4             .printSchema()
5            )
```

▸ (1) Spark Jobs

```
root
 |-- date: string (nullable = true)
 |-- county: string (nullable = true)
 |-- state: string (nullable = true)
 |-- fips: string (nullable = true)
 |-- cases: string (nullable = true)
 |-- deaths: string (nullable = true)
```

CSV Example

- According to the peak, a few columns should be integer values
- We can add the “inferSchema” option to have spark infer the schema automatically

```
1  covidDF = (spark.read
2             .option("header","true")
3             .option("inferSchema","true")
4             .csv("/databricks-datasets/COVID/covid-19-data/us-counties.csv")
5             .printSchema()
6            )
```

▸ (2) Spark Jobs

```
root
 |-- date: string (nullable = true)
 |-- county: string (nullable = true)
 |-- state: string (nullable = true)
 |-- fips: integer (nullable = true)
 |-- cases: integer (nullable = true)
 |-- deaths: integer (nullable = true)
```

CSV Example

- Inferring the schema will take longer and use more resource
- Spark will have to read the data twice
  - Read data to infer schema
  - Read data again and map to the inferred schema

CSV Example

- Instead of inferring the schema, we can declare the schema ourselves
  - **Datatypes**: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html#data-types

Cmd 43

```
1  from pyspark.sql.types import *
2
3  covidSchema = StructType([
4    StructField("date", DateType(), True),
5    StructField("county", StringType(), True),
6    StructField("state", StringType(), True),
7    StructField("fips", IntegerType(), True),
8    StructField("cases", IntegerType(), True),
9    StructField("deaths", IntegerType(), True),
10 ])
```

```
1  covidDF = (spark.read
2             .option("header","true")
3             .schema(covidSchema)
4             .csv("/databricks-datasets/COVID/covid-19-data/us-counties.csv")
5             .printSchema()
6            )
```

```
root
 |-- date: date (nullable = true)
 |-- county: string (nullable = true)
 |-- state: string (nullable = true)
 |-- fips: integer (nullable = true)
 |-- cases: integer (nullable = true)
 |-- deaths: integer (nullable = true)
```

CSV Example

- We only want data from Orange county California.
- Nothing happens here because sort and filter are lazy

Cmd 4

```
1  (covid_df
2   .sort(covid_df["date"].desc())
3   .filter((covid_df["county"] == "Orange") & (covid_df["state"] == "California")))
```

```
Out[27]: DataFrame[date: string, county: string, state: string, fips: int, cases: int, deaths: int]
```

Transformations vs Actions in Spark

- Transformations are LAZY
  - Transformations are functions that produce new RDD/DF from existing RDDs/DFs
  - Transformations are executed only when we call an action
  - There are two types of transformations
    - Narrow
      - all the elements that are required to compute the records in single partition live in the single partition of parent RDD (map,filter,sample….)
    - Wide
      - the elements that are required to compute the records in the single partition may live in many partitions of parent RDD(Distinct,join,GroupByKey...)

Transformations vs Actions in Spark

- Actions are EAGER
  - Actions are operations that give non-RDD values
  - Values of action are stored to drivers or to the external storage system.
  - Examples
    - Count
    - Collect -> often used to check if the entire RDD can fit in the memory of the driver
    - reduce

	General	Math / Statistical	Set Theory / Relational	Data Structure / I/O
Transformations	map filter flatMap mapPartitions mapPartitionsWithIndex groupBy sortBy	sample randomSplit	union intersection subtract distinct cartesian zip	keyBy zipWithIndex zipWithUniqueID zipPartitions coalesce repartition repartitionAndSortWithinPartitions pipe
Actions	reduce Collect head show aggregate fold first take foreach top treeAggregate treeReduce foreachPartition collectAsMap toLocalIterator	count takeSample max min sum histogram mean variance stdev sampleVariance countApprox countApproxDistinct	takeOrdered	saveAsTextFile saveAsSequenceFile saveAsObjectFile saveAsHadoopDataset saveAsHadoopFile saveAsNewAPIHadoopDataset saveAsNewAPIHadoopFile

Cmd 5

```
1  display(covid_df
2   .sort(covid_df["date"].desc())
3   .filter((covid_df["county"] == "Orange") & (covid_df["state"] == "California")))
```

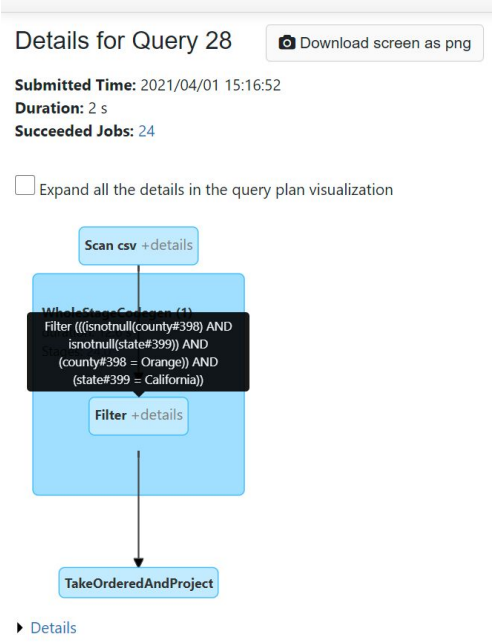
▼ (1) Spark Jobs
▼ Job 24 View (Stages: 1/1)
Stage 24: 4/4

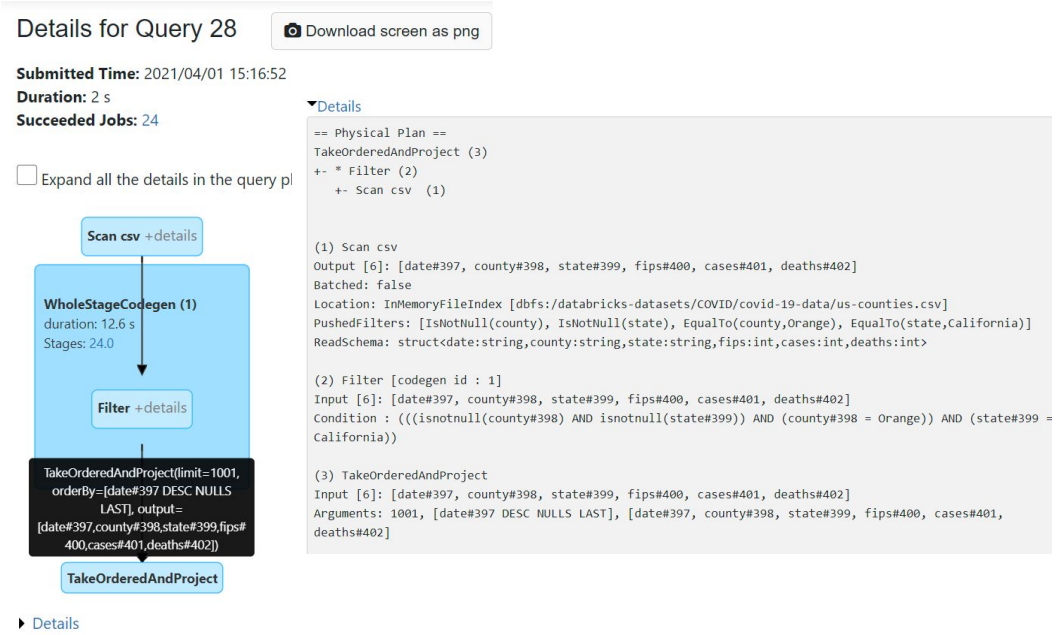
	date	county	state	fips	cases	deaths
1	2021-03-11	Orange	California	6059	263279	4379
2	2021-03-10	Orange	California	6059	263111	4346
3	2021-03-09	Orange	California	6059	262995	4313
4	2021-03-08	Orange	California	6059	262849	4252
5	2021-03-07	Orange	California	6059	262674	4226
6	2021-03-06	Orange	California	6059	262550	4173
7	2021-03-05	Orange	California	6059	262241	4075

Showing all 412 rows.

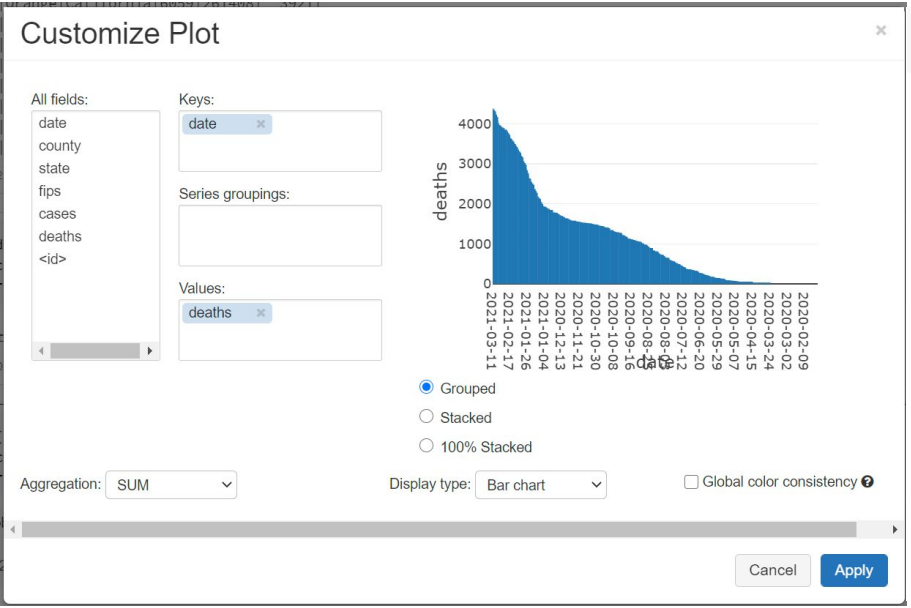
Automatic query optimization

- Sort and filter has been swapped during optimization
-





Antra



Antra

Reading JSON Data - inferSchema

- Similar to CSV data but with a few differences
  - No header, so only one job needed even when inferring the schema
  - Column names are extracted from JSON object attributes

Cmd 19

```
1  display(
2    spark.read
3    .option("inferSchema","true")
4    .json("/databricks-datasets/learning-spark-v2/blogs.json")
5    .printSchema()
6  )
```

▸ (1) Spark Jobs

```
root
 |-- Campaigns: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- First: string (nullable = true)
 |-- Hits: long (nullable = true)
 |-- Id: long (nullable = true)
 |-- Last: string (nullable = true)
 |-- Published: string (nullable = true)
 |-- Url: string (nullable = true)
```

Reading JSON Data - Predefined Schema

- Manually defining the schema can be a lot of work, might not be worth it for small files.
- For large files, this might save a lot of time spent on the infer-schema process.

Cmd 20

```
1  from pyspark.sql.types import *
2  jsonSchema= StructType([
3    StructField("Campaigns",ArrayType(StringType()),True),
4    StructField("First",StringType(),True),
5    StructField("Hits",LongType(),True),
6    StructField("Id",LongType(),True),
7    StructField("Last",StringType(),True),
8    StructField("Published",StringType(),True),
9    StructField("Url",StringType(),True)
10 ])
11
12 (spark.read
13   .schema(jsonSchema)
14   .json("/databricks-datasets/learning-spark-v2/blogs.json")
15   .printSchema()
16 )
```

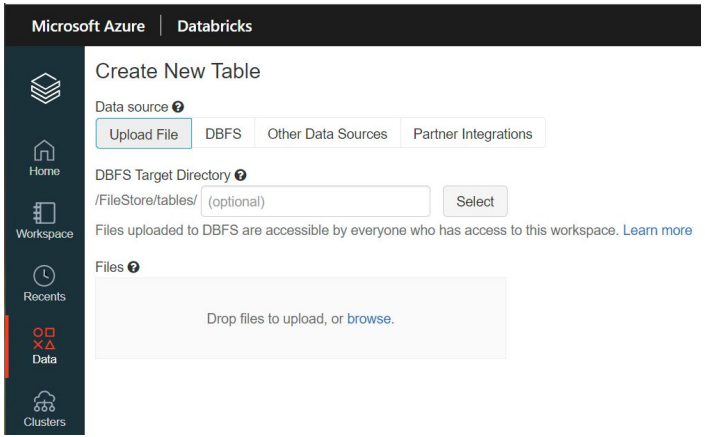
```
root
 |-- Campaigns: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- First: string (nullable = true)
 |-- Hits: long (nullable = true)
 |-- Id: long (nullable = true)
 |-- Last: string (nullable = true)
 |-- Published: string (nullable = true)
 |-- Url: string (nullable = true)
```

Reading Parquet Data

- Parquet is a column oriented storage format designed for big data
  - Optimized for reading and computing on columns
  - Metadata contains the schema
  - Compress better than row oriented storage
  - Easily splittable into multiple files
- Providing the schema when reading parquet data is not recommended
  - Reading in the schema from parquet metadata files is very cheap
  - Runtime exception if user provided schema is different from the schema stored in metadata.
- When reading, we provide the path to the directory that contains the parquet files instead of the path to individual files.

Reading Data using the UI

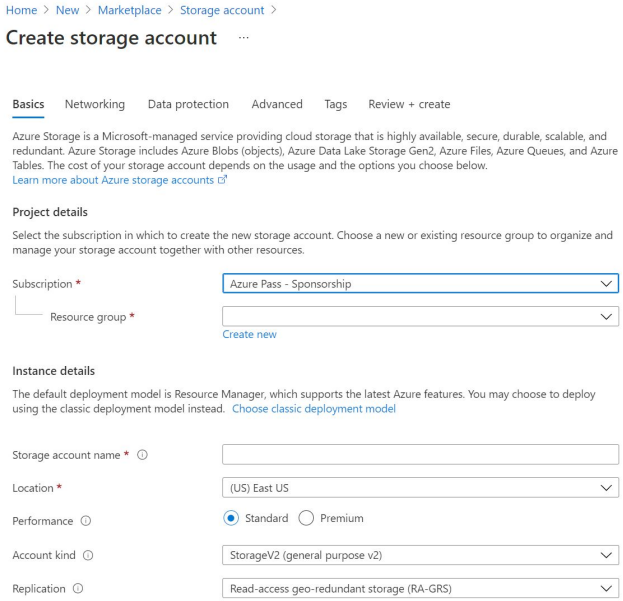
- We can use the “Data” tab in the UI to load “tables”
  - Once we create the table, it will stay in the data tab and we’ll never have to do it again
  - Available for any users with the right permission
  - Users will not see the credentials used to load the table



Antra

Demo-Upload file to azure blob and access from databricks

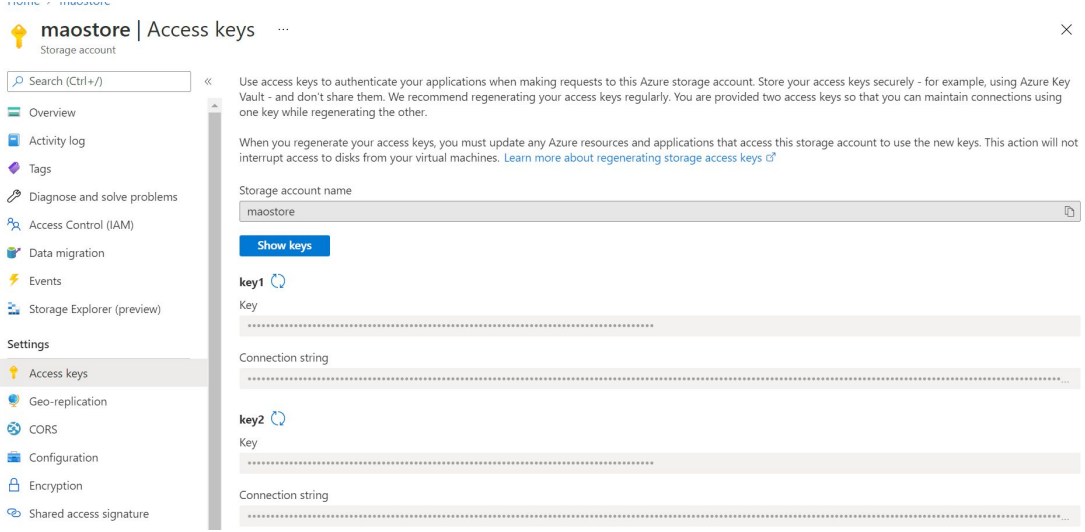
- Start with creating a storage account
- Select general purpose v2 and enable ADLS Gen2



Antra

Demo

- Go to “Access keys” to grab a connection string and store as a environmental variable.



Antra

Demo

- Run **pipenv install azure-storage-blob** in terminal
- Create a blob service client
- Either create a container or get a existing container client
- Get a blob client
- ...

Creating Tables from DataFrames

- Creating tables will allow us to query using Spark SQL
  - **df.createOrReplaceTempView("table_name")** can be used to create a temp table available only in the current notebook.
  - **df.createOrReplaceGlobalTempView("table_name")** can be used to create temp tables available in other notebooks

```
parquetDF.createOrReplaceGlobalTempView("parquet_table")
```

Cmd 22

```
%sql
select * from parquet_table order by requests desc limit(5)
```

▸ (1) Spark Jobs

	project	article	requests	bytes_served
1	en	Main_Page	865692	0
2	en.m	Main_Page	176949	0
3	en	Special:Search	76231	0
4	en.m	Donald_Trump	59847	0
5	en	Midas	55210	0

Write Data in Azure Databricks

Antra

- Use **.write** on a DataFrame to write data to the DBFS

```
1  fileName = userhome + "/pageviews_by_second.parquet"
2  print("Output location: " + fileName)
3
4  (csvDF.write                               # Our DataFrameWriter
5    .option("compression", "snappy") # One of none, snappy, gzip, and lzo
6    .mode("overwrite")                       # Replace existing files
7    .parquet(fileName)                       # Write DataFrame to Parquet files
8  )
```