Understanding Role of Apache Spark and Databricks

What is Apache Spark?

- Apache spark is a unified, open source, parallel, data processing framework for Big Data Analytics
- It is open source, extremely fast, powerful analytics engine for large-scale data processing of any type.
- It has in memory engine, can run the workload 100x faster than Hadoop
- It is highly scalable architecture, allows to run hundreds of machines together (Cluster computing) and process terabytes of data in parallel.
- Spark is used in variety of Use cases-**batch processing, streaming data, machine learning, advanced analytics.**

Challenges associated with spark

- Infrastructure management
- Manual configuration
- Upgrades
- Other Tools required
- Lake of interface

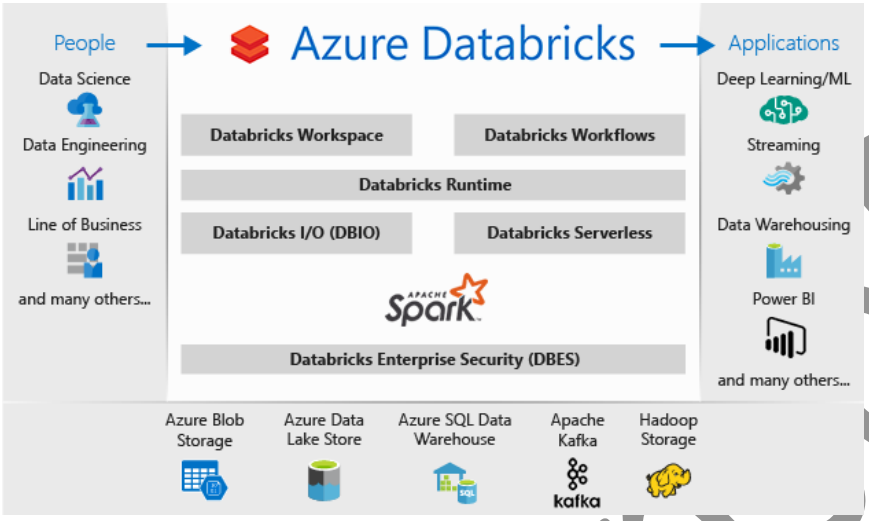
Databricks:

- **Databricks** is an organization and big data processing platform founded by the creators of Apache Spark.
- Databricks is managed and optimized platform for running Apache Spark, powered by processing capabilities of SPARK.
- Databricks was designed to unify data science, data engineering, and business data analytics on Spark by creating an easy to use environment
- It enables users to spend more time working effectively with data, and less time focused on managing clusters and infrastructure
- It has tools with it so you can quickly start building your application with spark.
- It provides intuitive UI and integrated workspace where you can work in collaboration with your colleague
- It allows you to setup and configure infrastructure with just few clicks.
- Scalability, failures, recovery, upgrades are manged by Databricks on its own.

Introduction to Azure Databricks

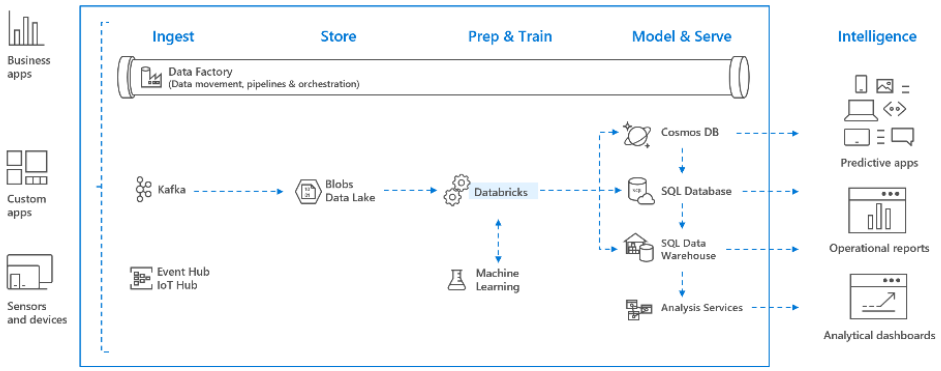
- Azure Databricks is Fully managed fast, easy highly Scalable, collaborative **Apache spark-based** analytics platform optimized for Azure.

- Databricks is integrated with Azure to provide simple,quick ,one-click setup,and streamlined workflows and enterprise scale and security of Microsoft's Azure platform.It has an interactive workspace.
- **Databricks workspace** is collaborative environment and you can explore data interactively and you can build **end to end workflow** by **orchestrating notebooks**, enables collaboration between data scientist, data engineers and Business Analysis.



- Databricks Meet the need of different users like consumers to data scientists
- **Data engineers** can use it to transform data and create and schedule **batch and streaming ETL** Jobs.
- **Data Scientist** can use it to create **machine learning modals** and other **analytics tasks**.
- **Business users** can write **SQL Queries** and **analyse and visualize** data using notebooks.
- It provides native integration with Azure services like SQL DW,Power BI,Blob storage,data lake store ,CosmosDB etc.
- Azure Databricks provides enterprise-grade Azure security, including Azure Active Directory integration, role-based controls giving fine-grained user permissions for notebooks, clusters, jobs, and data.

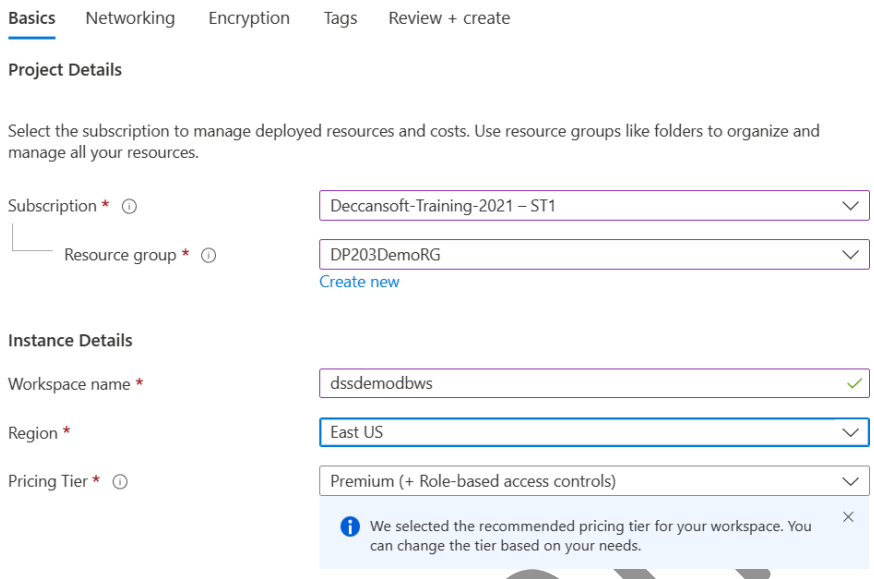
Data Pipeline and Azure Data Bricks:



Provision Azure Databricks

Lab 1: Provision Azure Databricks

Portal→Create Resource→Analytics→Databricks→



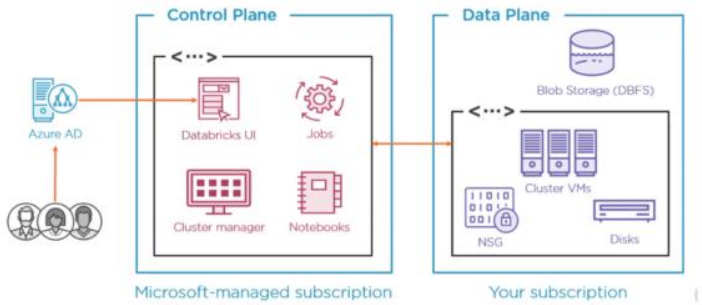
→review Create

Pricing Tier:

- **Standard** - Core Apache Spark capabilities with Azure AD integration.
- **Premium** - Role-based access controls and other enterprise-level features.
- **Trial** - A 14-day free trial of a premium-level workspace

Note: Just observe Manged Resource Group →Navigate →Observe the resources created in Data Plane

How databricks resources are deployed in Azure?



Workloads in Databricks:

Azure Databricks is optimized for three specific types of data workload and associated user persona

- Data Science and Engineering
- Machine Learning
- SQL*(Available in Premium tier)

Data Science and Engineering:

- Data engineers, data scientists, and data analysts can use interactive notebooks to run code in Python, Scala, SparkSQL, or other languages to **cleanse, transform, aggregate, and analyze data.**

Machine Learning:

- **Data exploration and preparation, training and evaluating** machine learning models, and serving models to generate predictions for applications and analyses.
- Data scientists and ML engineers can use AutoML to quickly train predictive models, or apply their skills with common machine learning frameworks such as SparkML, Scikit-Learn, PyTorch, and Tensorflow.
- They can also manage the end-to-end machine learning lifecycle with MLFlow.

SQL:

- This enables **data analysts to query, aggregate, summarize, and visualize data** using familiar SQL syntax and a wide range of SQL-based data analytical tools.

Databricks Key Concepts

1. **Apache Spark clusters** - Each Spark cluster has a *driver* node to coordinate processing jobs, and one or more *worker* nodes on which the processing occurs. Each node to operate on a subset of the job in parallel; reducing the overall time for the job to complete.
2. **Databricks File System (DBFS)** – Databricks has native support of distributed file system. File system is required to persist the data. When you create a cluster in databricks it comes with pre-installed DBFS. It is mounted into an Azure Databricks workspace and available on Azure Databricks clusters. One storage Account is mounted azure storage by default

   DBFS is just an abstraction layer and it uses at backend azure storage to persist the data, so users working with files store file in DBFS, but those files persist on Azure storage.

   Benefits:

   - Allows you to **mount** storage objects so that you can seamlessly access data without requiring credentials.
   - Allows you to interact with object storage using directory and file semantics instead of storage URLs.

- Persists files to object storage, so you won't lose data after you terminate a cluster.

Refer: https://docs.microsoft.com/en-us/azure/databricks/data/databricks-file-system

3. **Notebooks** - Notebooks provide an interactive environment where you can write code to work with Spark .You can combine text and graphics in *Markdown* format with cells containing code that you run interactively in the notebook session.
4. **Hive metastore** - *Hive* is an open source technology used to define a relational abstraction layer of tables over file-based data. The tables can then be queried using SQL syntax. The table definitions and details of the file system locations on which they're based is stored in the metastore for a Spark cluster. A *Hive metastore* is created for each cluster when it's created.
5. **Delta Lake** - *Delta Lake* builds on the relational table schema abstraction over files in the data lake to add support for SQL semantics commonly found in relational database systems. Capabilities provided by Delta Lake include transaction logging, data type constraints, and the ability to incorporate streaming data into a relational table.
6. **SQL Warehouses** – A SQL warehouse is a relational compute resource that lets you run SQL commands on data objects within Databricks SQL. The results of SQL queries can be used to create data visualizations and dashboards to support business analytics and decision making. SQL Warehouses are only available in *premium* tier Azure Databricks workspaces.

Introduction to Databricks Data Science and Engineering Environment

Workspace:

- A **workspace** is an environment for accessing all of your Azure Databricks assets. A workspace organizes objects (notebooks, libraries, dashboards, and experiments) into **folders** and provides access to data objects and computational resources.

Notebook:

- A web-based interface to documents that contain runnable commands, visualizations, and narrative text.

Repo:

- A folder whose contents are co-versioned together by syncing them to a remote Git repository.

Workflow:

- Azure Databricks job can be used to run a data processing or data analysis task.Job can be single task or can be a large, multi-task workflow. You can implement job tasks using notebooks, JARS, Delta Live Tables pipelines, or Python, Scala, Spark submit, and Java applications.

Data(Database and Tables):

- An Azure Databricks database is a collection of tables. An Azure Databricks table is a collection of structured data.

- It is created using files lying in storage.
- You can query using Spark API and Spark SQL or write to tables

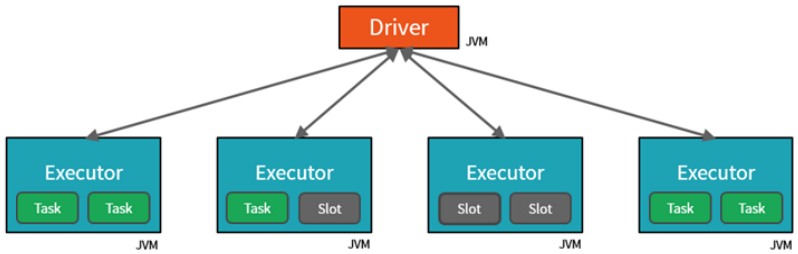
Note:Observe the URL which is unique for each workspace.It has the format
adb-<workspace-id>.<random-number>.azuredatabricks.net.
https://adb-2533046352263436.16.azuredatabricks.net/?o=2533046352263436#

Cluster

Apache Spark Cluster

- Apache Spark clusters are groups of computers that are treated as a single computer and handle the execution of commands issued from notebooks.
- They consist of a Spark *driver* and *worker* nodes. The driver node sends work to the worker nodes and instructs them to pull data from a specified data source.
- In Databricks, the notebook interface is typically the driver program and creates distributed datasets on the cluster, then applies operations to those datasets.
- Driver programs access Apache Spark through a *SparkSession* object.

Understanding Spark Job:



1. Work submitted to the cluster is split into as many independent jobs as needed.
2. Each job is broken down into *stages* to ensure everything is done in the right order
3. Stages are further subdivided into tasks
4. By splitting the work into tasks, the driver can assign units of work to **slots** in the executors on worker nodes for parallel execution. Additionally, the driver determines how to partition the data, assigns a partition of data to each task

Spark parallelizes jobs at two levels:

- The first level of parallelization is the *executor* - a Java virtual machine (JVM) running on a worker node, typically, one instance per node.

- The second level of parallelization is the ***slot*** - the **number** of which is determined by the **number of cores and CPUs** of each node.
- Each executor has multiple slots to which parallelized tasks can be assigned.

Azure Databricks Cluster:

An Azure Databricks cluster is a set of computation resources and configurations on which you run notebooks and jobs.

There are two types of clusters:

1. **Interactive(All Purpose):** You create an *interactive cluster* using the UI, CLI, or REST API. Multiple users can analyse the data together interactively.
2. **Automated (Job):** The Azure Databricks job scheduler creates *an automated cluster.*

Interactive Clusters	Automated(Job) Clusters
For collaborative interactive ad-hoc analysis,data exploration or developement	For automated jobs
Created by user .Users can share it.	Auto created when job starts
Can be manually terminated and restarted	Terminates when job ends.Cannot be restarted.
Option is provided to Auto terminate ,if idle for specified amount of time	Auto terminate option is not applicable
Low Execution time	Overhead involved to start cluster with job but Provides high throughput as all resources are dedicated for particular job
Can autoscale on demand	Can autoscale on demand
Costly than Job Clusters	Comparatively cheaper

Note: *Azure Databricks retains cluster configuration information for up to 200 interactive clusters terminated in the last 30 days and up to 30 automated clusters recently terminated by the job scheduler. To keep an interactive cluster configuration even after it has been terminated for more than 30 days, an administrator can* ***pin*** *a cluster to the cluster list.*

Interactive Cluster Modes: Standard, High Concurrency,Single Node

1. **Standard Mode:** Recommanded for Single User workloads that require multiple worker nodes,Supports Scala,SQL,Python,R and Java.
2. **Single Node** :Intended for jobs that use small amounts of data or non-distributed workloads such as single-node machine learning libraries.
3. **High Concurrency:** Ideal for groups of users who need to share resources or run ad-hoc jobs.

Note:

- *Standard mode clusters* are now called ***No Isolation Shared access mode clusters*.**
- *High Concurrency with Tables ACLs* are now called ***Shared access mode clusters*.**
- **Access control list (ACL):**A list of permissions attached to the workspace, cluster, job, table, or experiment. An ACL specifies which users or system processes are granted access to the objects, as well as what operations are allowed on the assets. Each entry in a typical ACL specifies a subject and an operation.

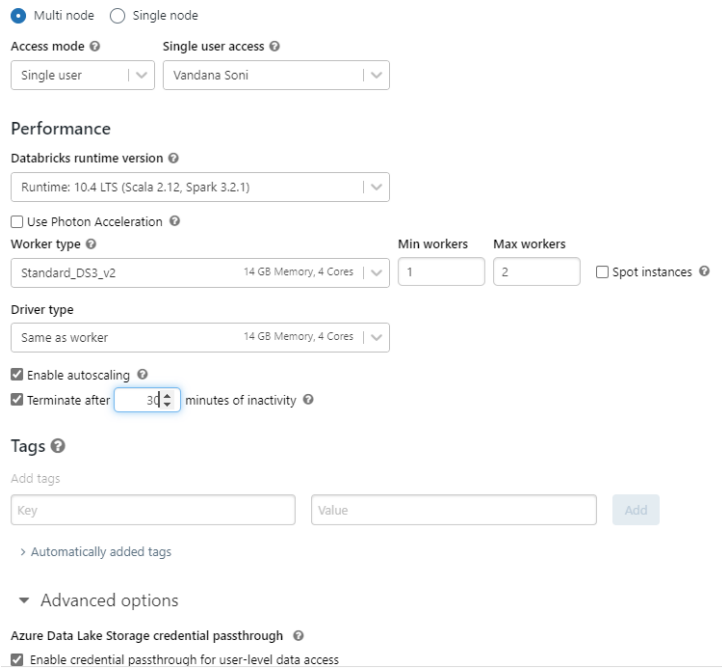
For new Cluster UI Refer: *https://learn.microsoft.com/en-us/azure/databricks/clusters/cluster-ui-preview*

New Cluster UI:

Original cluster settings	New access mode
Standard	No isolation shared
Standard + credential passthrough checkbox checked	Single user + credential passthrough checkbox checked
High concurrency (HC)*	No isolation shared
HC + Table Access Control checkbox checked	Shared
HC + Credential Passthrough checkbox checked	Shared + credential passthrough checkbox checked

Lab 2: Create Cluster

Clusters Tab→+Create Cluster→



→Create Cluster

Cluster policies: are a set of rules used to limit the configuration options available to users when they create a cluster. Cluster policies have ACLs that regulate which specific users and groups have access to certain policies. **Personal Compute** is a Azure Databricks-managed **default cluster policy** available on all Azure Databricks workspaces. The policy allows users to easily create single-machine compute resources for their individual use so they can start running workloads immediately, minimizing compute management overhead.

Refer:https://learn.microsoft.com/en-us/azure/databricks/clusters/personal-compute

Databricks runtime version:

It is VM image,comes with pre-installed libraries,have specific version of spark,scala and other libraries. Azure Databricks offers several types of runtimes.You can select based on your workloads need.

- Databricks Runtime includes Apache Spark but also adds a number of components that run on Databricks clusters which substantially improve the usability, performance, and security of big data analytics.
- When you set up a cluster you need to select version of data bricks runtime which comes bundled with
    - Runtime Specific Apache spark version
    - Additional set of optimizations
    - In azure it runs on Ubuntu OS,so it comes with system libraries of UBunto.
    - All the languages with corresponding libraries preinstalled
- Databricks Runtime ML is a variant of Databricks Runtime that adds multiple popular machine learning libraries, including TensorFlow, Keras, PyTorch, and XGBoost.
- Databricks Light provides a runtime option for jobs that don't need the advanced performance, reliability, or autoscaling benefits provided by Databricks Runtime.

- Azure Databricks supports clusters accelerated with graphics processing units (**GPUs**) ,For computationally challenging tasks that demand high performance, like those associated with deep learning

Note:You can edit the settings and change cluster configuration but it may require restart of cluster.

Worker Type and Driver Type:

- These are different VM Sizes provided by AZURE. Based on your requirement of memory, Cores and hard disk you can select it.
- Worker nodes actually perform data processing task. Having more worker nodes helps in faster processing as data is processed in parallel
- Driver node is responsible for taking request, distributing task to worker node and co-ordinating execution.
- All the runtime libraries will be installed on each worker nodes and driver nodes.

DBU: Databricks units is unit of processing capability per hour. You will be billed for no of DBU's Consumed .

Autoscaling: Cluster can scale on demand between specified minimum and maximum no of worker nodes.

Auto -Termination: Terminates interactive cluster if there is no activity for specified no of minutes.

Pools:

- Azure Databricks **pools reduce cluster start and auto-scaling times** by maintaining a set of idle, ready-to-use instances.
- Multiple clusters can be attached to a pool.
- When a cluster attached to a pool needs an instance, it first attempts to allocate one of the pool's idle instances.
- If the pool has no idle instances, the pool expands by allocating a new instance from the instance provider in order to accommodate the cluster's request.
- After cluster terminates , it releases an instance, it returns to the pool and is free for another cluster to use. Only clusters attached to a pool can use that pool's idle instances.
- Azure Databricks does not charge DBUs while instances are idle in the pool. Instance provider billing does apply

Pool Properties:

- **Idle Instance Auto Termination:** Setting in minutes to terminate the idle instance if it is idle for specified amount of time.
- **Minimum Idle Instances:** The minimum number of instances the pool keeps idle. These will always be available in pool regardless of Auto Termination specified.

- **Maximum Capacity:** setting to define upper limit of instances in pool(idle+used). If a cluster using the pool requests more instances than this number during autoscaling, the request will fail
- **Instance types:** Clusters attached to a pool use has to use the same instance type(configuration) for the driver and worker nodes

Notebook

- A notebook is a web-based interface to a document that contains runnable code, visualizations, and narrative text.
- Databricks notebooks provide real-time coauthoring in multiple languages, automatic versioning, and built-in data visualizations.
- **A notebook is a collection of runnable cells (commands). These cells are run to execute code, to render formatted text, or to display graphical visualizations.**
- You can execute a single cell or notebook at once. The notebook must be attached to a cluster. If the cluster is not running, the cluster is started when you run one or more cells.
- **Revision History**:You can see all the changes made to code.You can restore to any previous version.
- **Magic command**:Any command starting with % is magic command.
- **Command comments:** You can have discussions with collaborators using command comments.

%<language>	You can override the default language by using this at beginning of cell. Ex:%python,%sql
%fs	Allows you to use dbutils filesystem commands
%md	Markdown is a popular way of adding formatting.It is useful for various types of documentation, including text, images, and mathematical formulas and equations.

You can use Apache Spark notebooks to:

- Read and process huge files and data sets
- Query, explore, and visualize data sets
- Join disparate data sets found in data lakes
- Train and evaluate machine learning models
- Process live streams of data
- Perform analysis on large graph data sets and social networks

Lab 3: Working With Notebook

1. Create notebook

Note: *Notebook is created by selecting some language and that language is shown as default language.You can change it*

2. Create cells with following commands

"Hello Databricks"
x = 35 y = x + 5 print(y)
%scala var no=1+1 no=no+1
%md ### Hello This is title **This is Bold** __This is also Bold__ *This is italics*
x = 35 y = x + 5 print(y)

Databricks File System (DBFS)

- Databricks has native support of distributed file system. File system is required to persist the data. When you create a cluster in databricks it comes with pre-installed DBFS.
- It is mounted into an Azure Databricks workspace and available on Azure Databricks clusters. One storage Account is mounted azure storage by default
- DBFS is just an abstraction layer and it uses at backend azure storage to persist the data, so users working with files store file in DBFS, but those files persist on Azure storage.

Benefits:

- Allows you to **mount** storage objects so that you can seamlessly access data without requiring credentials.
- Allows you to interact with object storage using directory and file semantics instead of storage URLs.
- Persists files to object storage, so you won't lose data after you terminate a cluster.

DBFS Root:

The default storage location in DBFS is known as the *DBFS root*. Several types of data are stored in the following DBFS root locations:

/FileStore: Imported data files, generated plots, and uploaded libraries.

/databricks-datasets: Sample public datasets

/user/hive/warehouse: Data and metadata for non-external Hive tables.

Enable DBFS File Browser:

To see Mount point in Databricks

1. Databricks workspace→Click Top right(User account)→Admin Console→Workspace settings→
2. Advanced:DBFS File browser: **Enabled**
3. Refresh page

Refer: https://learn.microsoft.com/en-us/azure/databricks/dbfs/

Lab 4:Upload Data To DBFS and Use Spark to analyse it.

1. Upload Sample Data To DBFS

   File→Upload Data To DBFS→Browse and upload Products.cvs file→Next→Copy code generated
2. Create a New Notebook→Paste the code→Run the cell

display(df1)
select + and then select **Visualization** to view the visualization editor, and then apply the following options: • **Visualization type**: Bar • **X Column**: Category • **Y Column**: *Add a new column and select* **ProductID**. *Apply the* **Count** *aggregation*.
df1.write.saveAsTable("products")
%sql SELECT ProductName, ListPrice FROM products WHERE Category = 'Touring Bikes';

To use Azure storage and Data Lake store you need to mount it to DBFS(file storage for databricks)

- Mounting storage account to DBFS provides semeless access to data without requiring credentials.
- Once mounted ,you can use file semantics instead of URLs

Mount Azure Blob storage containers to DBFS

- You can mount a Blob storage container or a folder inside a container to Databricks File System (DBFS). You can only mount block blobs to DBFS.
- Azure Blob storage can be mounted using Access Key or SAS.
- DBFS uses the credential that you provide when you create the mount point to access the mounted Blob storage container
- Once mounted instead of using URLs you can use file system semantics

- The mount is a pointer to a Blob storage container, so the data is never synced locally. As file persist on storage ,they are safe even if you delete the cluster or workspace.

Note:One Azure storage is mounted to DBFS by default when you create databricks workspace instance.

Lab 5:Upload Sample data to datalake account and Azure Storage

There are two ways to access Azure Blob storage: account keys and shared access signatures (SAS).

Python Command:

```
dbutils.fs.mount(
  source = "wasbs://<container-name>@<storage-account-name>.blob.core.windows.net",
  mount_point = "/mnt/<mount-name>",
  extra_configs = {"<conf-key>":dbutils.secrets.get(scope = "<scope-name>", key = "<key-name>")})
```

Note:

<conf-key> can be

- fs.azure.account.key.<storage-account-name>.blob.core.windows.net
- fs.azure.sas.<container-name>.<storage-account-name>.blob.core.windows.net

Mount Azure Blob Storage

```
# Using Key
containername="taxidata"
storageaccountname="dssdemosa"
confkey="fs.azure.account.key."+storageaccountname+".blob.core.windows.net"
```

```
dbutils.fs.mount(
  source = "wasbs://"+containername+"@"+storageaccountname+".blob.core.windows.net",
  mount_point = "/mnt/taxidata",
  extra_configs = {confkey:
"rMwHb0/u0N7LwJUoY+j6VZg8u5qiRfiyxceUnQfyibCiBkk7hK6A2wTHVrXKIroNa/xigJ/Gv3LEytY0VtE75Q=="})
```

```
display(dbutils.fs.ls("/mnt/taxidata"))
```

```
#Using SAS
confkeysas
="fs.azure.sas."+storage_container_name+"."+storage_account_name+".blob.core.windows.net"
sasstr="sp=r&st=2021-09-02T16:01:14Z&se=2021-09-03T00:01:14Z&spr=https&sv=2020-08-04&sr=c&sig=LQEPi%2F6AbiyCYg7QQTgvQUM0q4rDBphTo%2F2cqiikJIQ%3D"
```

```
dbutils.fs.mount(
  source = "wasbs://"+containername+"@"+storageaccountname+".blob.core.windows.net",
```

```
  mount_point = "/mnt/taxidata1",
  extra_configs = {confkeysas:
"rMwHb0/u0N7LwJUoY+j6VZg8u5qiRfiyxceUnQfyibCiBkk7hK6A2wTHVrXKIroNa/xigJ/Gv3LEytY0VtE75Q=="})
```

```
display(dbutils.fs.ls("/mnt/taxidata"))
```

```
dbutils.fs.unmount("/mnt/taxidata1")
```

Lab 6:Mount Azure Blob storage containers to DBFS

Refer:03_MountStorage.ipynb

Accessing Azure Data Lake Gen2

There are four ways of accessing Azure Data Lake Storage Gen2:

1. Pass your Azure Active Directory credentials, also known as **credential passthrough.**
2. Mount an Azure Data Lake Storage Gen2 filesystem to DBFS using a service principal and OAuth 2.0.
3. Use a service principal directly.
4. Use the Azure Data Lake Storage Gen2 storage account access key directly.

What is Credential Passthrough?

- You can authenticate automatically **Azure Data Lake Storage** from Azure Databricks clusters using the same Azure Active Directory (Azure AD) identity that you use to log into Azure Databricks
- In cluster you can enable **Azure Data Lake Storage credential passthrough.**
- It allows**,** commands that you run on that cluster to read and write data in Azure Data Lake Storage without requiring you to configure service principal credentials for access to storage.

Use following command:

```
configs = {
"fs.azure.account.auth.type": "CustomAccessToken",
"fs.azure.account.custom.token.provider.class":
spark.conf.get("spark.databricks.passthrough.adls.gen2.tokenProviderClassName")
}
# Optionally, you can add <directory-name> to the source URI of your mount point.
dbutils.fs.mount(
source = "abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/",
mount_point = "/mnt/<mount-name>",
extra_configs = configs)
```

```
configs = {
  "fs.azure.account.auth.type":  "CustomAccessToken",
  "fs.azure.account.custom.token.provider.class":
spark.conf.get("spark.databricks.passthrough.adls.gen2.tokenProviderClassName")
}

# Optionally, you can add <directory-name> to the source URI of your mount point.
dbutils.fs.mount(
  source = "abfss://taxidata@dssdemodatalake2.dfs.core.windows.net/",
  mount_point = "/mnt/taxidata3",
```

```
    extra_configs = configs)
```

```
dbutils.fs.ls("/mnt/taxidata3")
```

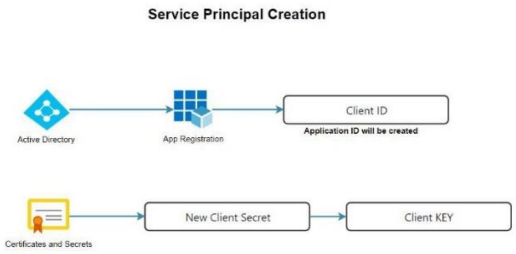
Mount an Azure Data Lake Storage Gen2 filesystem to DBFS using a service principal and OAuth 2.0.

Service Principal:

- It is an Identity which can be used by application to access Azure Resources.
- With applications use Service Principal Instead of using user's AD credentials.
- For this you need to register an application in Azure AD and create secret key which acts like password.
- The App id and secret can be used to access Azure resources.

Steps:

1. Create Service principal and make note of **appid** and **secret** and **directory id**:



Note the following properties:

- application-id: An ID that uniquely identifies the application.
- directory-id: An ID that uniquely identifies the Azure AD instance.
- Secret Value:Password created for application

2. **Set ACLs correctly for a service principal**
   - Go to required container(Directory)→Right click→Manage ACL→Configure access list and default list for ACL
3. **Use following Commands**

```
configs = {"fs.azure.account.auth.type": "OAuth",
          "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
          "fs.azure.account.oauth2.client.id": "<application-id>",
          "fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope = "<scope-name>", key =
"<key-name-for-service-credential>"),
          "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/<directory-
id>/oauth2/token" }
//Replace the values for "<application-id>,<directory-id> and specify secret value for client secret
```

```
configs = {"fs.azure.account.auth.type": "OAuth",
       "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
       "fs.azure.account.oauth2.client.id": "5b3856a8-3dd4-4c60-b729-91ea6de640db",
       "fs.azure.account.oauth2.client.secret": "gk07Q~3L6k1sLBeqQXb.jVgGWcxs8fXy0zXkt",
       "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/2de8d54d-5576-
4bf6-b419-6065cb1e700e/oauth2/token"}
#Replace the values for "<application-id>,<directory-id> and specify secret value for client secret
```

```
dbutils.fs.mount(
  source = "abfss://taxidata@dssdemodatalake2.dfs.core.windows.net/",
  mount_point = "/mnt/taxidata1",
  extra_configs = configs)
```

```
dbutils.fs.unmount("/mnt/taxidata1")
```

Use Access Key:

Use Access Key

```
containername="taxidata"
storageaccountname="dssdemodatalake"
confkey="fs.azure.account.key."+storageaccountname+".dfs.core.windows.net"
source = "abfss://"+containername+"@"+storageaccountname+".dfs.core.windows.net"
mountpoint="/mnt/taxidata1"
```

```
dbutils.fs.mount(
  source = source,
  mount_point = mountpoint,
  extra_configs = configs)
```

```
dbutils.fs.ls("/mnt/taxidata1")
```

Use the Azure Data Lake Storage Gen2 storage account access key directly.

```
storage_account_name="dssdemodatalake1"
storage_account_access_key="Ka7Kzf2+pGvCTuxzBAFVrcmK71rX+NHz60NmSd17EZZUEEG6xqq2LN/GwFLcosZfFDniPhlEMw67+A
```

```
spark.conf.set(
  "fs.azure.account.key."+storage_account_name+".dfs.core.windows.net",\
  storage_account_access_key)
```

```
dbutils.fs.ls("abfss://taxidata@dssdemodatalake.dfs.core.windows.net")
```

```
spark.read.csv("abfss://taxidata@dssdemodatalake.dfs.core.windows.net/Employee.txt").collect()
```

Lab7:Mount Azure Data Lake to DBFS

Refer: 03_MountStorage.ipynb

Read Data and Apply Schema

Read file into DataFrame

- PySpark supports reading files in CSV, JSON, and many more file formats into PySpark DataFrame
- Using csv("path") or format("csv").load("path") of DataFrameReader, you can read a CSV file into a PySpark DataFrame.
- You can use multiple options.You can chain options.

Examples:

```
df1 =spark.read.csv("/mnt/taxidata1/1000_Sales_Records.csv")
```

```
df2 = spark.read.option("header",True) \
    .csv("/mnt/taxidata1/1000_Sales_Records.csv")
```

```
df1.show(10)
```

```
display(df1)
```

```
df2.printSchema()
```

```
df3 = spark.read.options(header='True', inferSchema='True', delimiter=',') \
 .csv("/tmp/resources/zipcodes.csv")
```

```
df = spark.read.format("csv") \
 .option("inferSchema", "true") \
 .option("header", "true") \
 .option("sep", ",") \
 .load("/mnt/taxidata1/1000_Sales_Records.csv")
```

```
//Read Multiple Files
df = spark.read.csv("path1,path2,path3")
```

```
//Read all files in directory
df = spark.read.csv("Folder path")
```

Refer: https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/DataFrameReader.html

Applying Schema:

- PySpark **SQL Types** class is a base class of all data types in PySpark, defined in a package pyspark.sql.types.DataType.
- Spark DataFrames schemas are defined as a collection of typed columns. The entire schema is stored as a StructType and individual columns are stored as StructFields
- PySpark provides from *pyspark.sql.types import StructType* class to define the structure of the DataFrame. StructType is a collection or list of StructField objects.
- It provides *pyspark.sql.types import StructField class* to define the columns which includes column name(String), column type (DataType), nullable column (Boolean) and metadata (MetaData)

Lab 8:Read Data and Apply Schema

Import notebooks :04_TaxidataDemo.dbc

```
#Create own schema
#from pyspark.sql import SQLContext
#from pyspark.sql.types import StructType,StructField, StringType, IntegerType
from pyspark.sql.types import *
#from pyspark.sql.types.DataType import *
salesSchema = StructType([\
 StructField("Region",StringType(),True),
 StructField("Country",StringType(),True),
 StructField("ItemType",StringType(),True),
 StructField("SalesChannel",StringType(),True),
 StructField("OrderPriority",StringType(),True),
 StructField("OrderDate",TimestampType(),True),
 StructField("OrderID",IntegerType(),True),
 StructField("ShipDate",TimestampType(),True),
 StructField("UnitsSold",IntegerType(),True),
 StructField("UnitPrice",DoubleType(),True),
 StructField("UnitCost",DoubleType(),True),
 StructField("TotalRevenue",DoubleType(),True),
 StructField("TotalCost",DoubleType(),True),
 StructField("TotalProfit",DoubleType(),True)
])
```

```
#Attach Schema to DataFrame
salesDF = spark.read.format("csv")\
  .option("header","true")\
  .schema(salesSchema)\
  .load("/mnt/myfile/1000_Sales_Records.csv")
```

```
# Create schema for FHV Bases
from pyspark.sql.types import *
fhvBasesSchema = StructType([\
 StructField("License Number", StringType(), True),
 StructField("Entity Name", StringType(), True),
 StructField("Telephone Number", LongType(), True),
 StructField("SHL Endorsed", StringType(), True),
 StructField("Type of Base", StringType(), True),
 StructField("Address",
  StructType([\
   StructField("Building", StringType(), True),
   StructField("Street", StringType(), True),
   StructField("City", StringType(), True),
   StructField("State", StringType(), True),
   StructField("Postcode", StringType(), True)]),
   True),
 StructField("GeoLocation",
   StructType([\
   StructField("Latitude", StringType(), True),
   StructField("Longitude", StringType(), True),
   StructField("Location", StringType(), True)]),
  True)
 ])
```

```
fhvBaseDF = spark.read.json("/mnt/taxidata1/FhvBases.json")
```

```
fhvBaseDF = spark.read.option("multiline","true").json("/mnt/taxidata1/FhvBases.json")
```

```
#Add .schema(fhvBasesSchema)

fhvBaseDF1 = spark.read.format("json")\
.option("multiline","true")\
.load("/mnt/taxidata1/FhvBases.json")
```

Data Cleansing and Transformations

Data Quality Checks

1. Completeness
   - Remove null or missing data
   - Fill missing value with placeholder
2. Uniquness

- Remove Duplicate records

3. Timeliness
   - Appropriate date range
4. Accuracy
   - Remove inaccurate data

Transformations:

In spark you have similar types of transformation as in SSIS or INFORMATICA or ant other ETL tool.

In spark you can use any language of your choice to build ETL pipeline

Transformation	Spark	SQL
Filter Data	dataframe.where(<expression>	SELECT ….WHERE <expression>
Calculated Columns	Dataframe .withColumn("<col>",<expression>)	SELECT <expression> AS <col name>
Count Rows	Dataframe .count()	SELECT Count(*)
Aggreagte Data	Dataframe .groupBy(<columns to group on >) .agg(<aggregationexpression>)	SELECT <aggregation expression> GROUP BY <columns to group on>
Sort Dataset	Dataframe .orderBY(<columns to sort on>)	SELECT…. ORDER BY <columns to sort on>
Join 2 Datasets	Dataframe1 .join(dataframe2,<join condition> ,<join type>	SELECT … FROM table1 <join type> JOIN table2 <join condition>
Union	dataframe1 .union(dataframe2)	SELECT … FROM table1 UNION SELECT …FROM table2

Lab 9:Perform Transformations on GreenTaxi and Yellow Taxi Data

Refer:ETL Notebook

Execution Context:

- Execution context is isolated environment in which code is executed and state is maintained(variables,objects,functions)
- In Databricks,each language and notebook combination has separate execution context on cluster.
- Objects in one execution context cannot be shared with other execution context.
- Example:variables created in one language can't be used in other language or variable from one notebook cant be used in another.To work with multiple languages and notebooks , ,you need to pass around data from one execution context to another.

View:

- A view stores the text for a query typically against one or more data sources or tables in the metastore.
- In Databricks, a view is equivalent to a Spark DataFrame persisted as an object in a database.
- **Local temporary views** are scoped to the notebook or script level. They cannot be referenced outside of the notebook in which they are declared, and will no longer exist when the notebook detaches from the cluster.
- **Global temporary views** are scoped to the cluster level and can be shared between notebooks or jobs that share computing resources

Load Transformed Data to Files

There are multiple file formats in which we can load the data.

Example: csv, Json, Parquet etc

```
mydf
.write
```

```
.option("header","true")
.option("dateFormate","yyyy-MM-dd  HH:mm:ss.S")
.csv("/mnt/test/destination/airport_out.csv")
```

```
poutputpath="/mnt/test/parquetoutput/airport_out.parquet  "
mydf.write.format("parquet").save(poutputpath)
```

```
#Use savemode to Overwrite or Append
greentaxidf.write.format("parquet").mode("overwrite").save(poutputpath)
```

```
greentaxidf.write.mode("append").parquet(poutputpath)
```

Write property

- Write property on dataframe is object of class DataFrameWriter and has methods to write to external storage systems, by providing various options.
- **mode** can be used to set writing behaviour.
  Example : .mode(SaveMode.Overwrite)

SaveMode: There are four options

1. Append:Append contect to existing data
2. ErrorIfExists :Throw error if already exists
3. Ignore:Do not change existing data
4. Overwrite:Overwrite existing data

Partitioning

- Multiple files are created as a result of write operation
- RDDs/Dataframes are divided into multiple partitions and partitions are distributed to memory of multiple nodes in a cluster.
- Each node can have multiple partitions. Processing happens on partitions and parallelism is achieved.
- When you save the file instead of file, folder is created and each partition is written as separate file in it.
- Default no of partition for spark is 200
- Spark job or other tools with spark connector can read all these partitioned files by specifying folder name.
- Default partitions can be changed using spark.sql.shuffle.partitions configuration setting
- Use **coalesce or repartition** methods to change partitions on a DataFrame. Changing it results in movement of data between partitions.
- Rule of thumb: Use repartition() to increase partitions and coalesce() to decrease partitions.

Note:Configuring very less partitions for Large dataset may cause bottle neck and memory exceptions

Configure Partitions

- **spark.conf.get** method is used to get any configuration setting in spark

  spark.conf.get("spark.sql.shuffle.partitions")
- **spark.conf.set** method is used to Change the no of partitions for new dataframes. Old dataframes will not be affected by this

  spark.conf.set("spark.sql.shuffle.partitions",10)

Note:

Single partition is fine with smaller dataset but should not be used for larger dataset

Changing partitions may result in movement of data. All data coming to one partition can become bottleneck and take lot of time to save

Or may lead to memory exception

Apache Parquet file format

- It is columnar storage format
- It supports complex nested data structure
- Stores schema in the file itself so you need not infer schema
- It supports efficient compression and encoding
- Parquet files are very small as compared to csv and are binary files.
- It can take more time to write to parquet then to Csv as writing in slow
- Query from it is extremely fast than Csv/Json

Working with Tables

Database and Tables:

An Azure Databricks database is a collection of tables. An Azure Databricks table is a collection of structured data.

You can query tables with Spark APIs and Spark SQL and perform supported dataframe operations on it.

- Each workspace has central Hive metastore,to store the metadata of tables i.e. schema of table.It is accessible by all clusters
- Underlying data can be in any format -Csv,Json,Parquet,RDBMS table etc.
- Once table is defined you can use it as SQL Table or Dataframe without providing credentials
- Table can be of two types:
  1. **Managed:** A *managed* table is defined without a specified location, and the data files are stored within the storage used by the metastore. Dropping the table not only removes its metadata from the catalog, but also deletes the folder in which its data files are stored.
  2. **Unmanaged:**An *external* table is defined for a custom file location, where the data for the table is stored. The metadata for the table is defined in the Spark catalog. Dropping the table deletes the metadata from the catalog, but doesn't affect the data files.

Managed Table	Unmanaged Table
Schema and data is managed by Spark	Only Schema is managed by Spark,stored in Hive metastore
Data is stored in DBFS	Data is stored in external location
Stored in Parquet format	Stored in underlying dataset format(csv,json,paraquet,rdbms table etc)
Dropping table deletes Schema and data from DBFS. Deleting workspace also removes data as DBFS is removed.	Dropping Table deletes only Schema,data is not deleted
Useful to persist staging data	Useful to persist processed data

Create Managed Table:

dataframe.write. saveAsTable("<example-table>")

Unmanaged Table:

dataframe.write.option('path', "<your-storage-path>").saveAsTable("<example-table>")