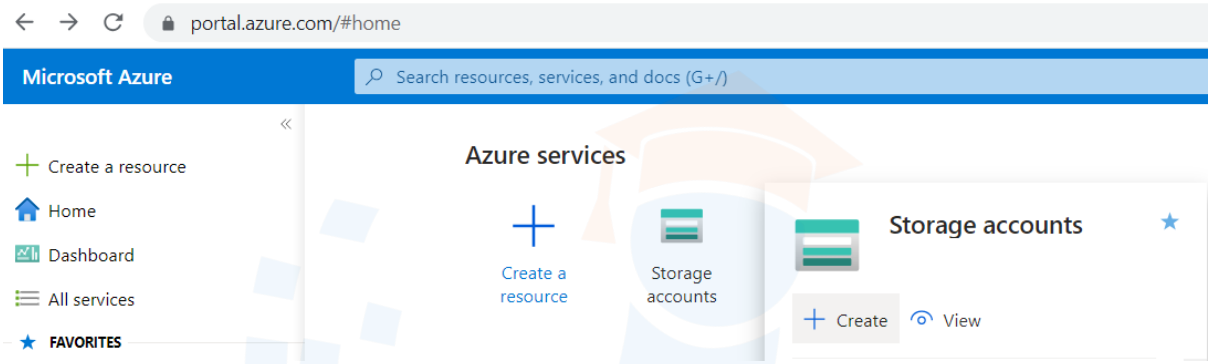
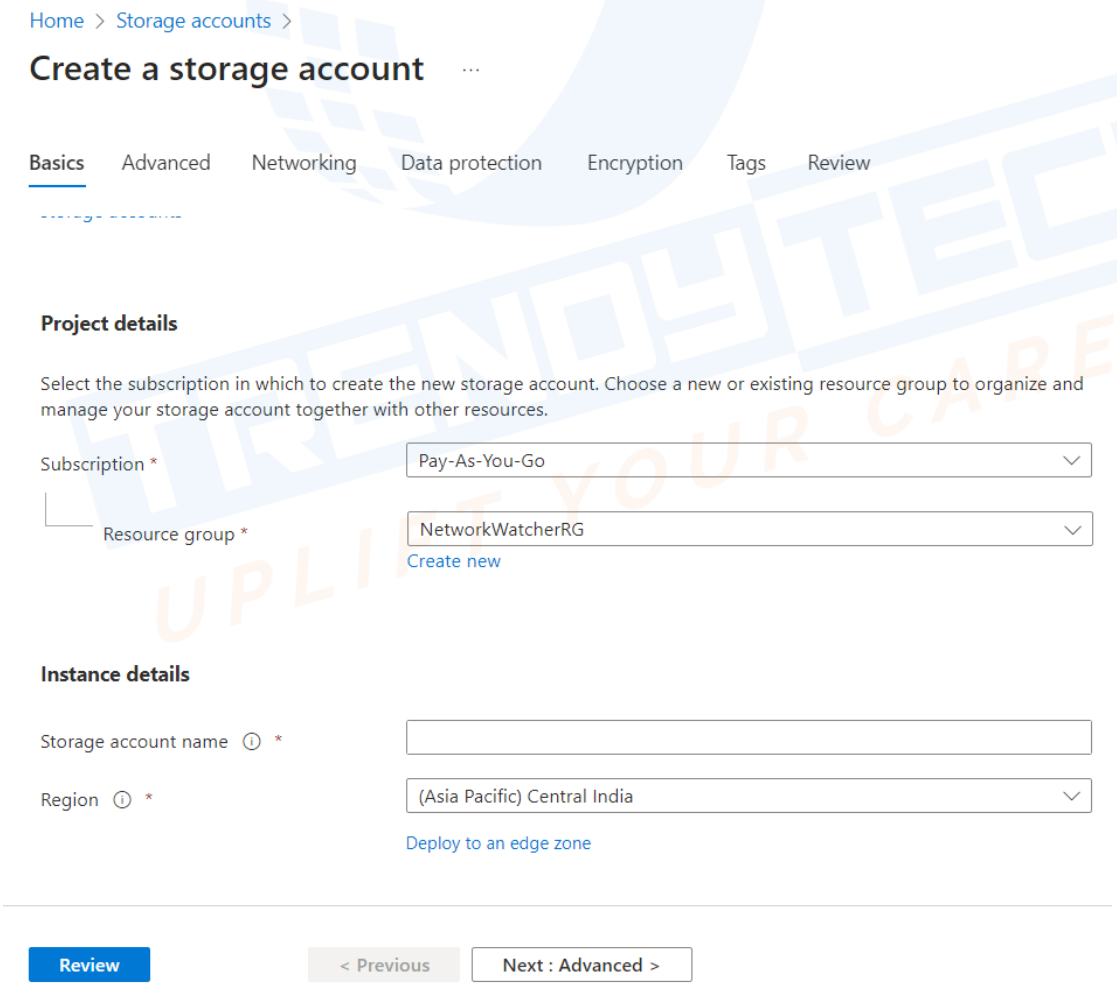
Azure Storage Account Service is all about storing the data on Azure Cloud.

This storage service is

- **Durable : Chances of services going down is negligible**
- **Scalable : Can handle huge amounts of increasing data(Petabytes)**
- **Secure : Uses Encryption Algorithms internally to maintain a high level of security.**





Different services in the Storage Account

1. BLOB / Containers Service - Binary Large Object

   It is a highly scalable object store and can hold any kind of data in its raw form. Containers refer to BLOB storage.

2. Table Service - Is like a NoSQL Database on Cloud. NoSQL tables that are semi-structured can be created using this service (Just like HBase).
3. File Share Service - Is used for mounting and storing files shares in case of lift & shift scenarios.
4. Queue Service - Is used for sending and receiving messages.

BLOB Access Tiers

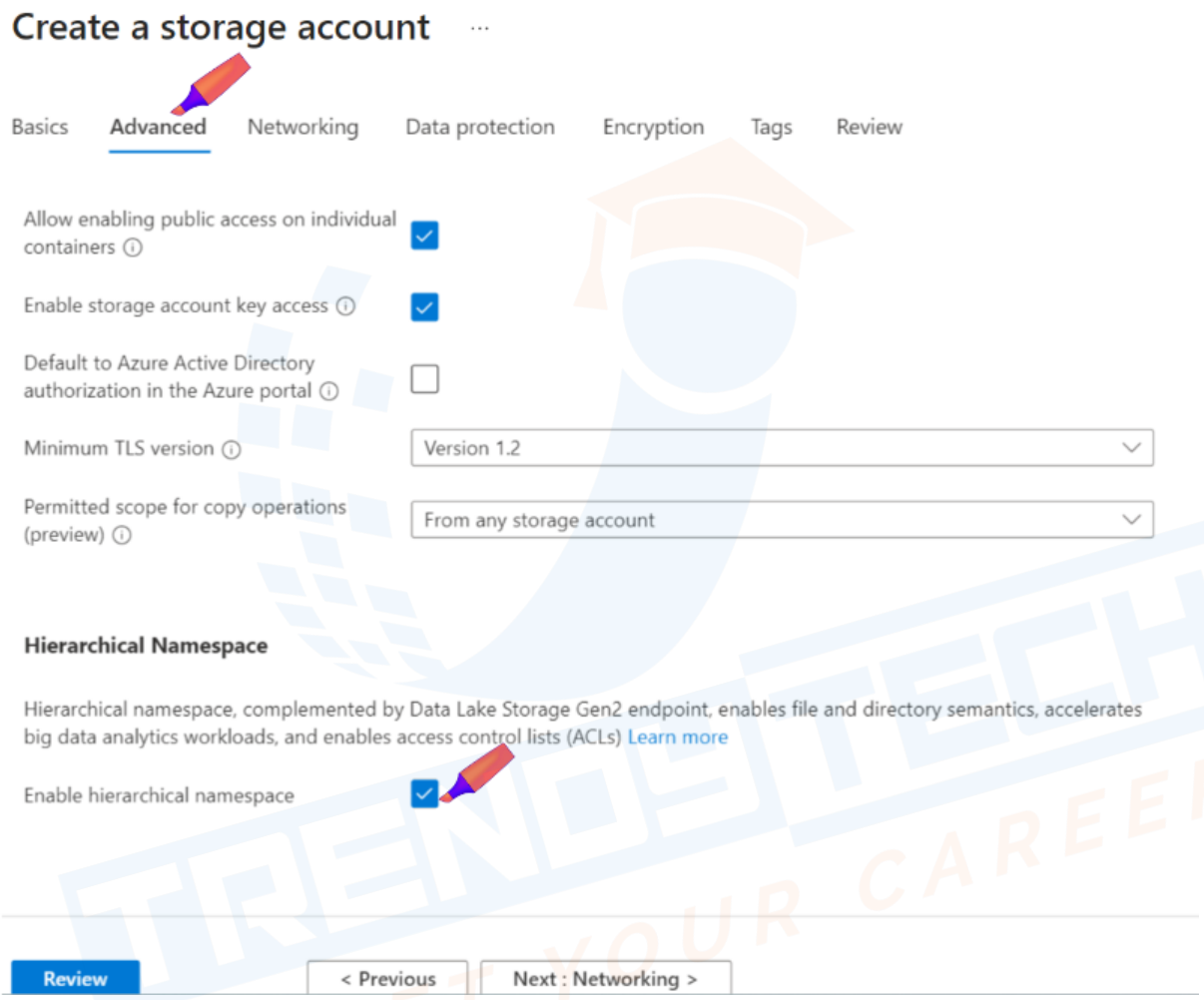
HOT	COLD	Archive
• For Frequently Accessed Data • Cost is High • High Availability and Performance	• For Infrequently Accessed Data • Cost is comparitively Less	• For Rarely Accessed Data • Cost is Low

Different Redundancy Options

Locally Redundant Storage-LRS	Zone Redundant Storage-ZRS	Geo Redundant Storage-GRS
• All the 3 copies of data are kept within the same Data Center on different Servers. • Protects fom Server / Rack Failures	• Data is replicated synchronously across 3 Availability Zones. • Protects from Data Center Level Failures	• Data Replicated to other Regions • Protects from Region Level Failures. • Example: Central India(LRS)⟶ South India(LRS)
Read Access Geo Redundant Storage-GRS	**Geo-Zone Redundant Storage-GRS**	**Read Access Geo-Zone Redundant Storage-GRS**
• Data can be accessed from both the Regions at all times. (Even if there was no region failure)	• Data Replicated to other Regions • Example: Central India(Across 3 Zones)⟶ South India(LRS)	• Data Replicated to other Regions • Data can be accessed from both the Regions at all times. (Even if there was no region failure)

Azure DataLake Storage Gen 2

Azure Datalake Storage Gen 2 account creation is similar to the Blob/Container Storage creation. However, the hierarchical namespace option has to be enabled under the advanced settings tab during the storage account creation.



Normal BLOB Storage	Azure Datalake Gen 2
Data storage	Data storage
Containers	Containers
File shares	File shares
Queues	Queues
Tables	Tables
• Hierarchial Namespace Disabled • Flat Heirarchy • Can contain only Files • No option to create sub directories • Doesn't support ACL	• Hierarchial Namespace Enabled • Allows Heirarchial Structure • Can contain sub-folders • Can create sub directories • Supports ACL(Access Control List)

Key Points

1. By default the access tier is hot
2. Archive is available at blob level. As we can archive only the existing data.
3. Changing the access tier from cold/cool -> hot (An early deletion charge will be applied if done within 30 days of storage creation)
4. Changing the access tier from archive -> hot/cool (An early deletion charge will be applied if done within 180 days of storage creation)
5. Rehydration is a process that is followed when moving the Blob from archive to hot/cool tier. This process takes several hours.

Storage Cost	Retrieval Cost
Hot Access Tier - Highest	Hot Access Tier - Lowest
Cool Access Tier - Medium	Cool Access Tier - Medium
Archive - Lowest	Archive - Highest

6. Life Cycle Management of Storage Account - Can be automated
7. Azure Storage Pricing
    a. Volume of data stored per month
    b. Quantity and types of operations performed along with data transfer cost if any.

What is Databricks?

- Databricks is an offering on-top of Apache Spark. (A company by the creators of Apache Spark)
- Apache spark based unified analytics platform optimised for cloud.

Challenges of Apache Spark Open Source Version

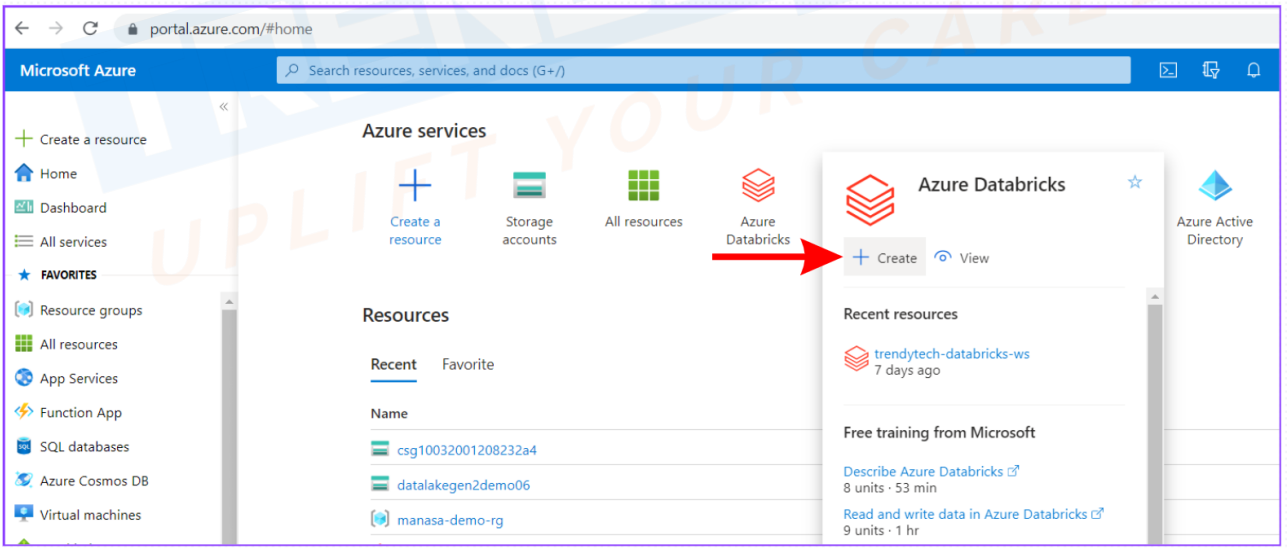
1. Infrastructure Management - Like procuring the VMs…
2. Software Installation
3. Upgrade and Maintenance Challenges
4. Lack of User Interface
5. Managing Security
6. Version Compatibility Clashes

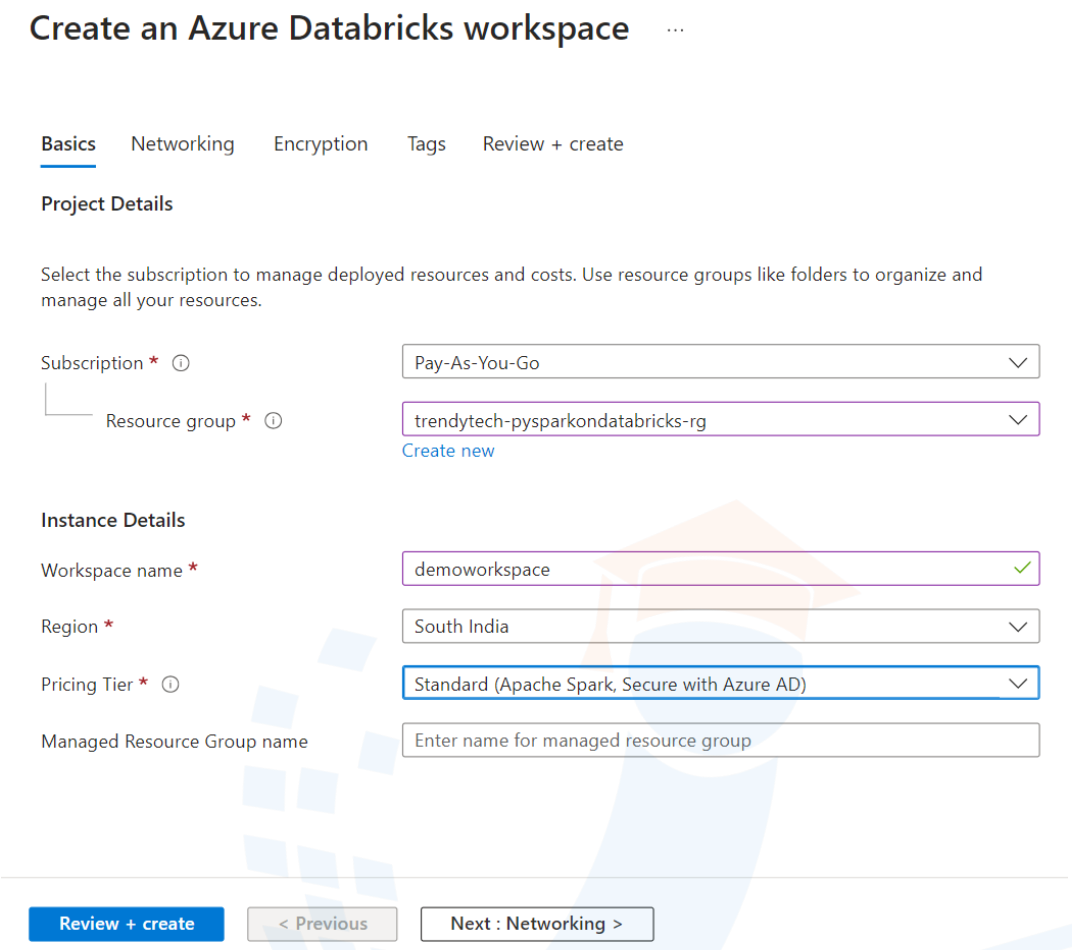
Why DataBricks?

- Provides solutions to all of the above mentioned challenges of open source spark.

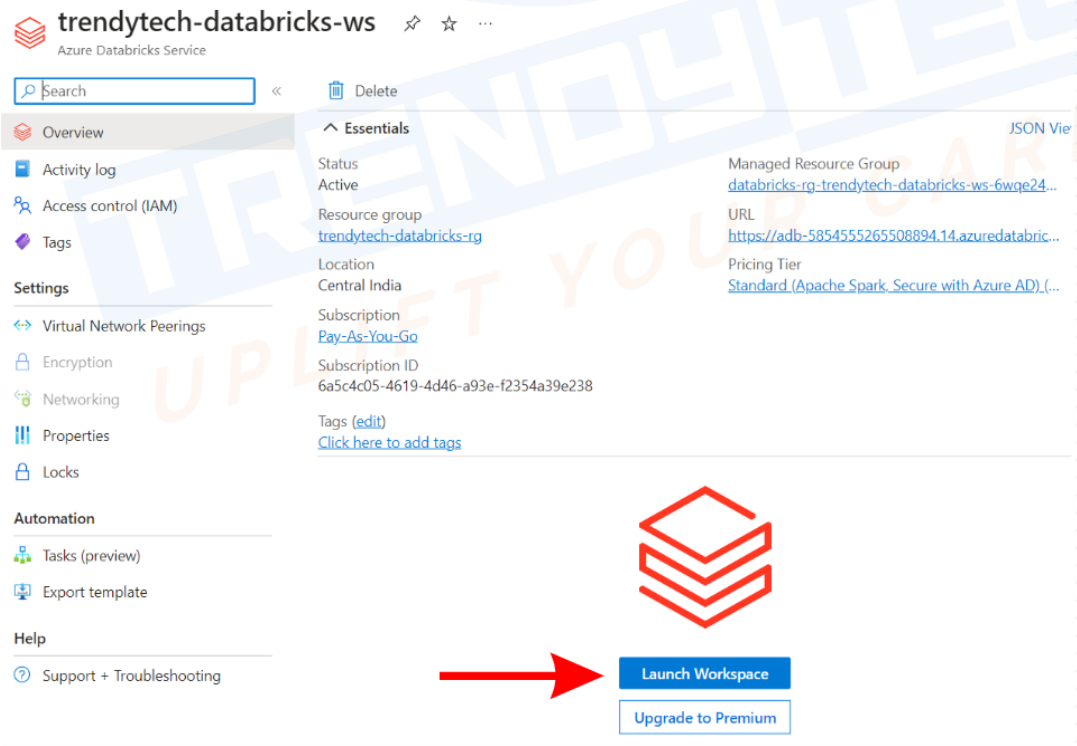
  Databricks sets up a cluster by procuring all the required resources with all the softwares installed on a click of a button. It also takes care of security, software upgrades and version compatibility along with a user-friendly UI.

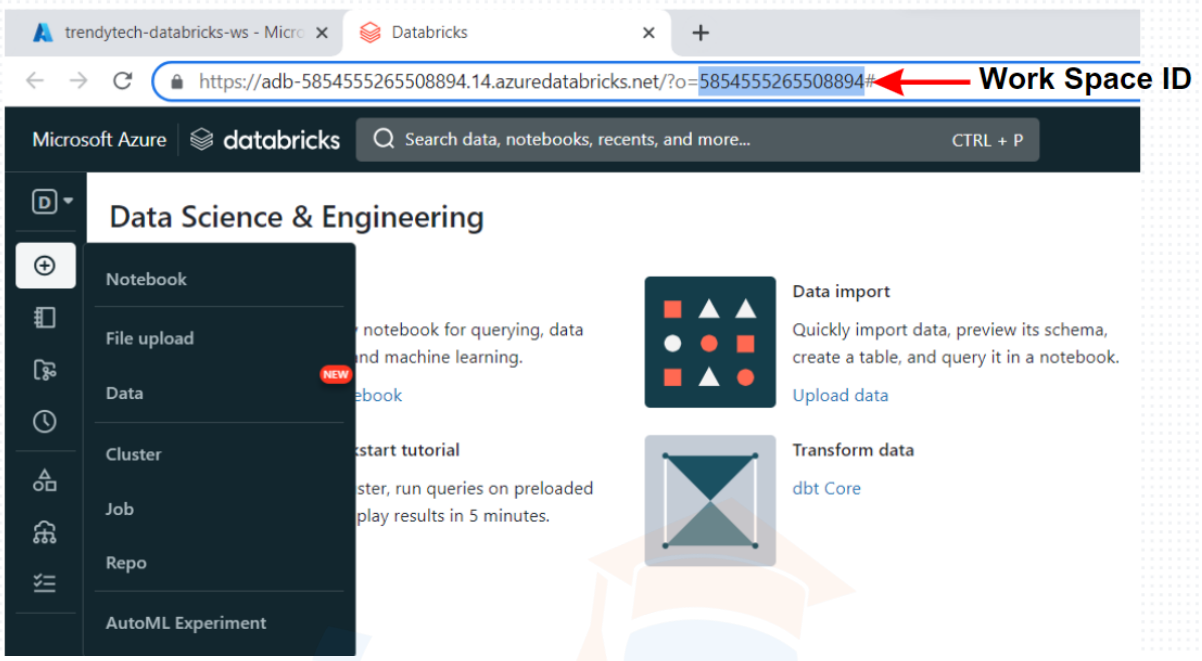
Creating Azure Databricks Instance / Workspace





Launching Databricks Workspace





Databricks Pricing

1. Infrastructure / Hardware Charges
2. Software Charges

Azure Databricks Pricing

Azure Infrastructure/ Hardware Charges

Consider an Example
Requesting for 4 Machines with
4 CPU Cores & 4GB RAM

Infrastructure Charges:
0.5$ per hour / Machine
Total = 0.5 * 4 = 2$ per hour

Databricks Software Charges

Software Charges:
0.5 DBU per Machine = 2 DBUs
(If 1 DBU = 0.5$)
Total = 0.5 * 2 = 1$

Note :

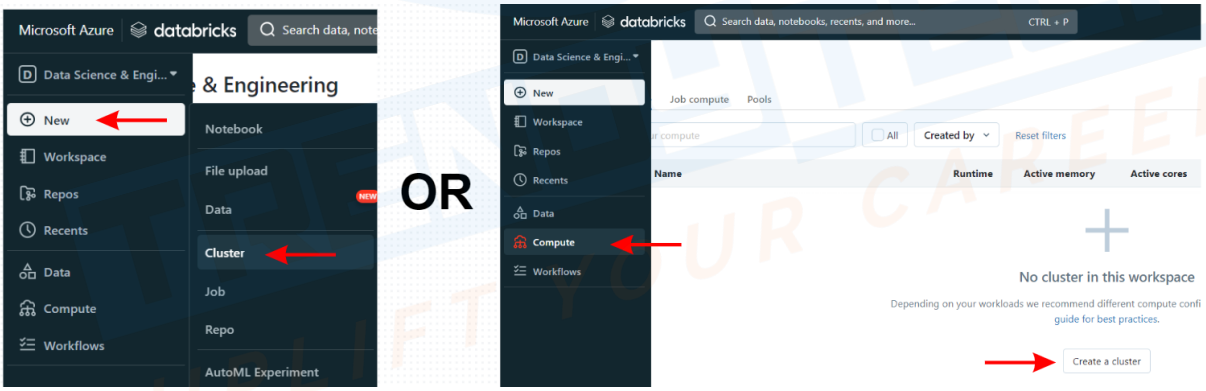
- Databricks is available across major cloud service providers - Azure, AWS, GCP
- Databricks is a first party managed service on Azure. Therefore, high quality support is given by Azure for Databricks service. That is the reason Azure + Databricks is in-demand in the industry.

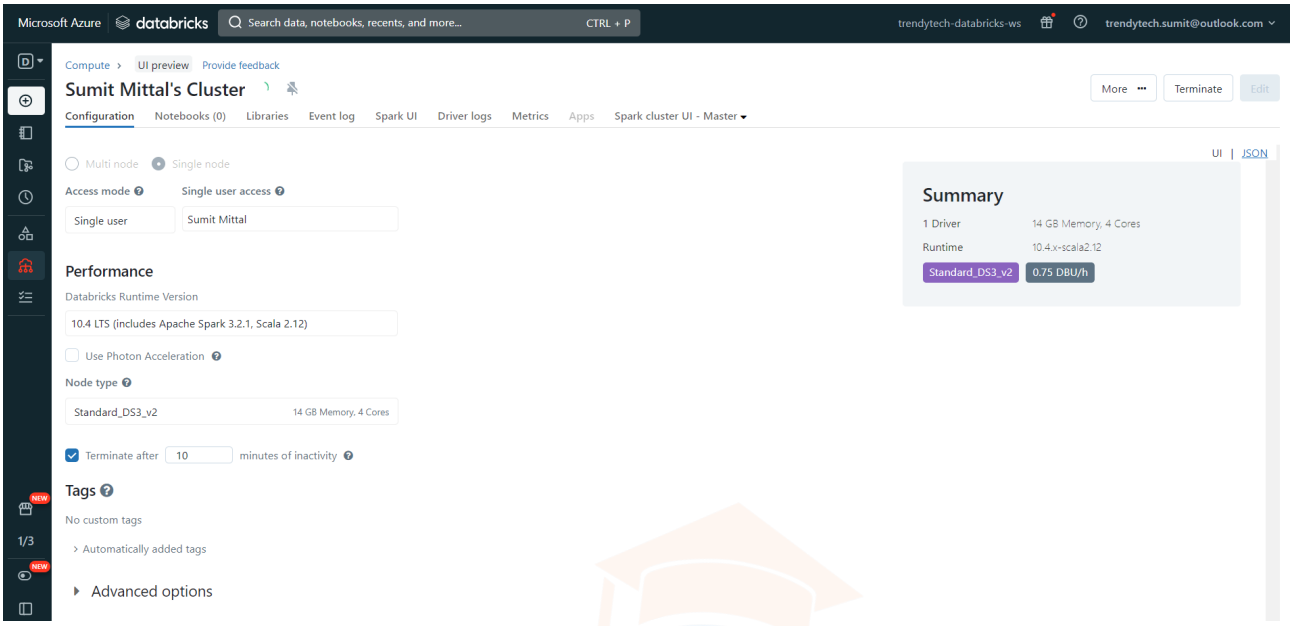
Cluster is a collection of VMs with Spark installed. Has two components

1. **Driver Node**
2. **Worker Node**

NoteBook - Is where the code is written/developed and executed for the development environment.

Creating Cluster





Cluster Creation Options

1. **All Purpose Cluster -** Cluster mostly used for interactive purposes
2. **Job Cluster -** Cluster created for a scheduled job and terminated after the job is executed
3. **Pool**

All Purpose Cluster	Job Cluster
• Created Manually • Persistent • Suitable for Interactive Workloads • Shared among many users • Expensive	• Created by Job • Terminated after the Job is complete • Suitable for Automated Workloads • Meant for a specific job and therefore isolated • Inexpensive

Cluster Modes

1. **Single Node -** Single node acts as both the driver and worker node.
2. **Standard -** Consists of a driver node and multiple workers.

(Both Single Node and Standard are meant for single user and cannot handle multiple users effectively)

3. **High Concurrency -** Consists of a driver node and multiple workers and can handle multiple users.

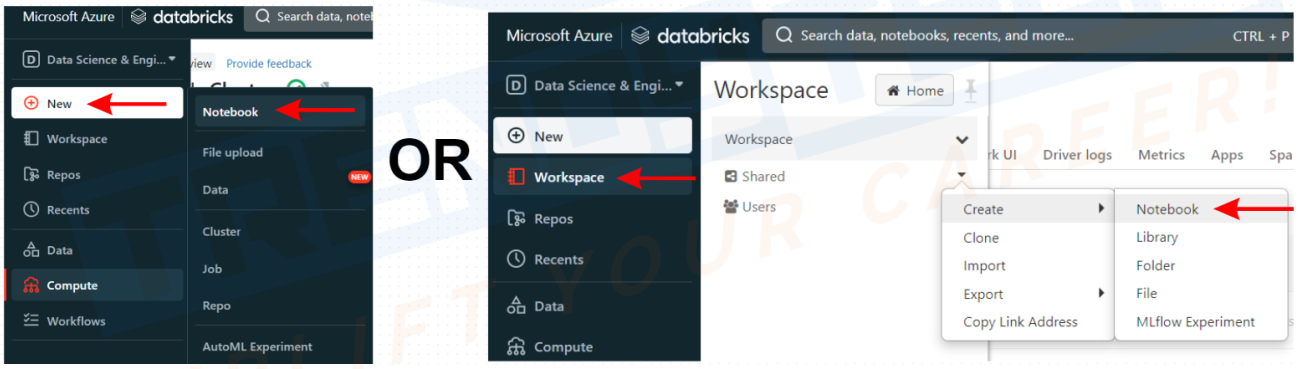
Azure Databricks Vs Azure Synapse

- More Optimized
- Latest Version of Spark available.
- Synapse also provides support for .net

Worker Node Types

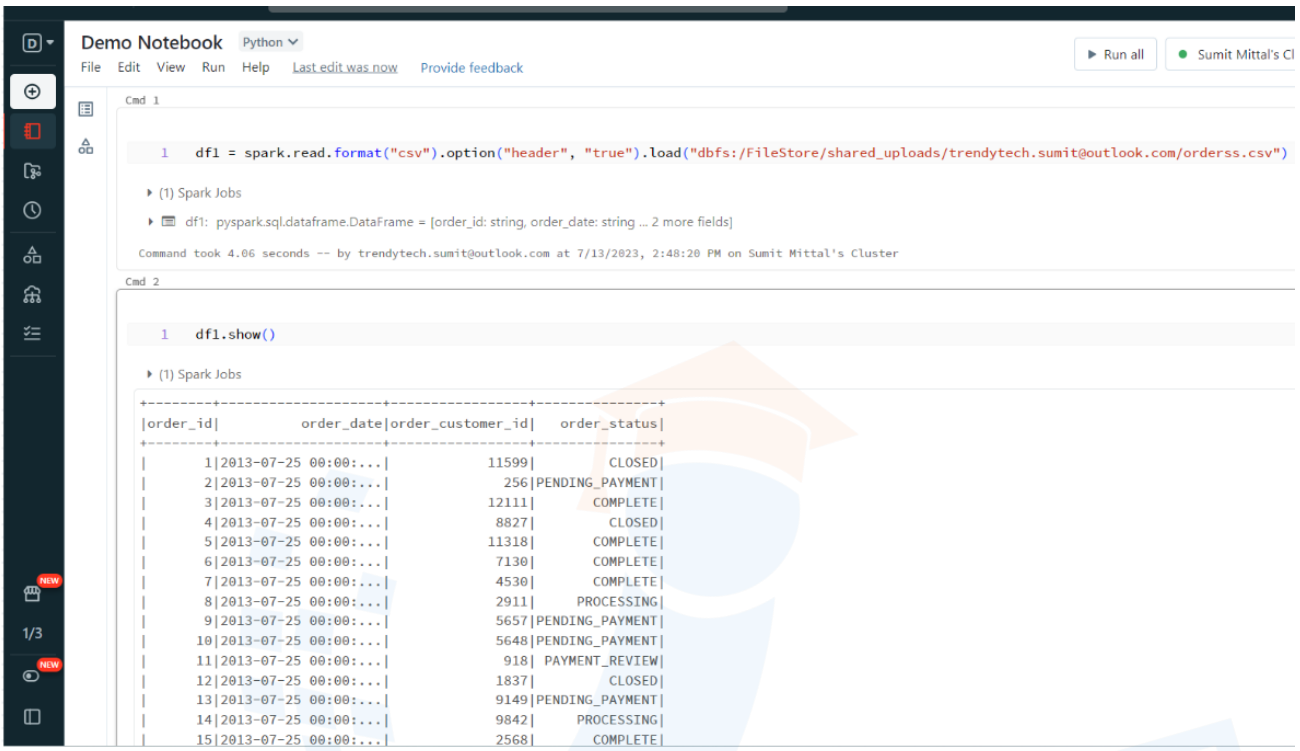
1. Memory Optimized - Meant for memory intensive operations like Machine Learning Workloads.
2. Compute Optimized - Meant for quick computation operations like Streaming Workloads.
3. Storage Optimized - For high disk throughputs.
4. General Purpose - Can be used for generic workloads.
5. GPU Accelerated - For deep learning

Creating a Notebook in the Cluster and executing sample code



- Select the language as Python and choose the cluster on which the code has to be executed.
- Upload the file from your local to Databricks File System using the file upload option.
- Create a dataframe and load the data from the file.
- It is possible to share the notebook with a team and work collaboratively.

- Magic commands are used to write various language codes in the same Notebook and for several other utility purposes.
- Auxiliary Magic Command %fs is to navigate the file system.



Databricks File System - DBFS

- DBFS is a wrapper on the azure storage like Blob, Datalake gen2.
- The actual data is stored in the Object Store (BLOB / Datalake gen2)
- Filestore is the DBFS root where the actual data is stored.

Architecture of Databricks

All the resources in Azure Databricks fall under 2 subscription planes

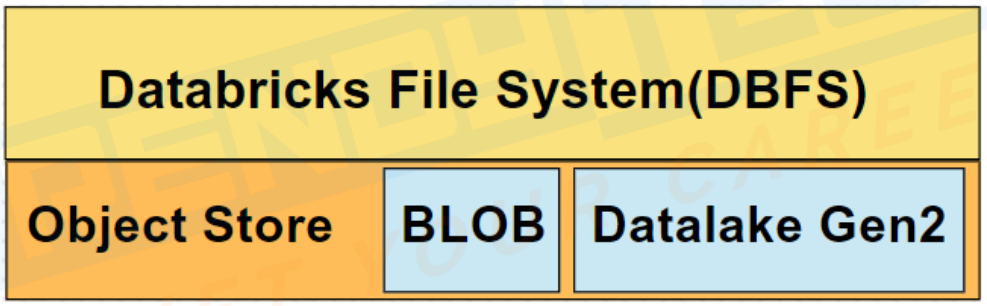
1. Control Plane - Whenever a databricks workspace is created, the resources that get deployed under the Databricks Subscription come under the control plane. These resources include - Databricks UI, Cluster Manager, DBFS, Cluster Metadata.
2. Data Plane - The resources that get deployed under your Azure Subscription come under the data plane. These resources include - VNET, NSG, Azure BLOB Storage.

Databricks Community Edition

- Databricks offers a community edition which is free of cost with limited access to resources.

Understanding DBFS in-depth

- DBFS is a distributed file system mounted into a databricks workspace.
- It is a wrapper / an abstraction on top of the scalable object stores like blob & azure datalake gen2.



- By default, DBFS file system browsing is disabled. To enable it, under Settings -> Admin Console -> Workspace Settings - > DBFS File Browser (has to be Enabled)

- **DButils** provides some utility functions to work on Databricks.

  dbutils.fs.help('cp') provides information about the **cp** command.

  dbutils.fs.ls('/') Lists all the files under root in DBFS.

dbutils.fs.head('<file-path>') Lists the first 65536 bytes of the file, present in the given file-path.

dbutils.fs.mkdirs('/FileStore/temp') Creates a directory named temp under FileStore.

dbutils.fs.mv('/FileStore/temp', '/FileStore/temp-new', True) Copies the file from source to destination and then deletes the file from source.

Data Utility

Create a Dataframe and pass it as a parameter to the summarize utility function

dbutils.data.summarize(df) Provides all the statistics of the data

```
df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/trendytech.sumit@outlook.com/orderss-1.csv")
dbutils.data.summarize(df)
```

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [order_id: string, order_date: string ... 2 more fields]

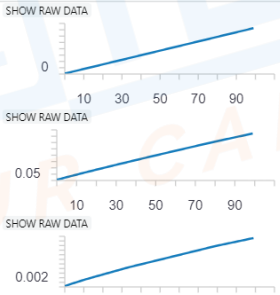
Sort by: Feature order | Reverse order | Feature search (regex enabled)

Features: string(4)

Categorical Features (4)

	count	missing	unique	top	freq top	avg len	custom
order_id	68.9k	0%	66.8k	"61649"	1	4.84	data type: string
order_date	68.9k	0%	353	2013-11-...	347	21	data type: string
order_customer_id	68.9k	0%	11.8k	"569"	11	4.11	data type: string

Chart to show: Standard | log | expand



Notebook Utility

dbutils.notebook.help() displays two functions

- exit (for exiting the notebook with a return value)
- run (required when the Notebooks are required to be executed in a sequence. This is performed using a Wrapper Notebook)

In order to initiate the execution of the child notebook from the wrapper notebook, use the following command

dbutils.notebook.run('<file-path-of-child-notebook>', '<Timeout-period>')

Widget Utility

dbutils.widgets.help() displays the following functions -

combobox, dropdown, multiselect, text

Example :

- **combobox**

//creating the widget combobox

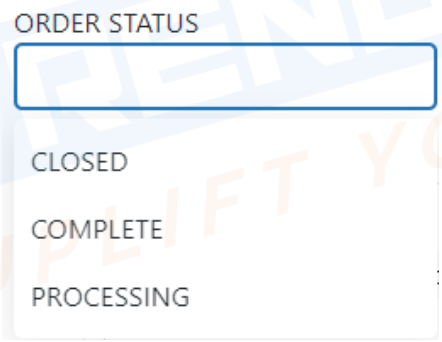
dbutils.widgets.combobox(name = 'orderstatus' , defaultValue = 'CLOSED' , choices = ['CLOSED', 'COMPLETE' , 'PROCESSING'] , label = 'ORDER STATUS')

//creating and loading a dataframe

df = spark.read.csv('<orders.csv-file-path>' , header = True)

//Accessing the dynamically added values in the widget combobox

os = dbutils.widgets.get('orderstatus')



//Displaying all the values from the dataframe that matches the dynamically added values in the combobox

df.where("order_status == '{}'".format(os)).show()

ORDER STATUS

CLOSED

Cmd 3

```
1   df1.where("order_status == '{}'".format(os)).show()
```

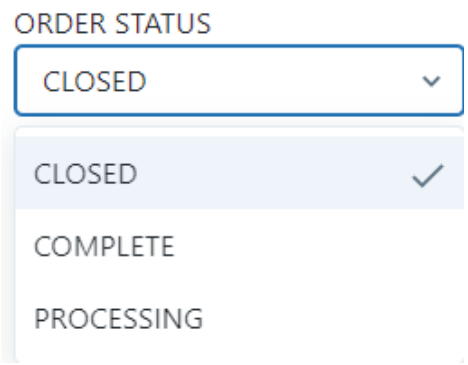
(1) Spark Jobs

```
+--------+--------------------+-----------------+------------+
|order_id|          order_date|order_customer_id|order_status|
+--------+--------------------+-----------------+------------+
|       1|2013-07-25 00:00:...|            11599|      CLOSED|
|       4|2013-07-25 00:00:...|             8827|      CLOSED|
|      12|2013-07-25 00:00:...|             1837|      CLOSED|
|      18|2013-07-25 00:00:...|             1205|      CLOSED|
|      24|2013-07-25 00:00:...|            11441|      CLOSED|
|      25|2013-07-25 00:00:...|             9503|      CLOSED|
|      37|2013-07-25 00:00:...|             5863|      CLOSED|
|      51|2013-07-25 00:00:...|            12271|      CLOSED|
|      57|2013-07-25 00:00:...|             7073|      CLOSED|
|      61|2013-07-25 00:00:...|             4791|      CLOSED|
|      62|2013-07-25 00:00:...|             9111|      CLOSED|
|      87|2013-07-25 00:00:...|             3065|      CLOSED|
|      90|2013-07-25 00:00:...|             9131|      CLOSED|
|     101|2013-07-25 00:00:...|             5116|      CLOSED|
|     116|2013-07-26 00:00:...|             8763|      CLOSED|
|     129|2013-07-26 00:00:...|             9937|      CLOSED|
|     133|2013-07-26 00:00:...|            10604|      CLOSED|
|     191|2013-07-26 00:00:...|               16|      CLOSED|
```

- **dropdown**

//creating the widget dropdown

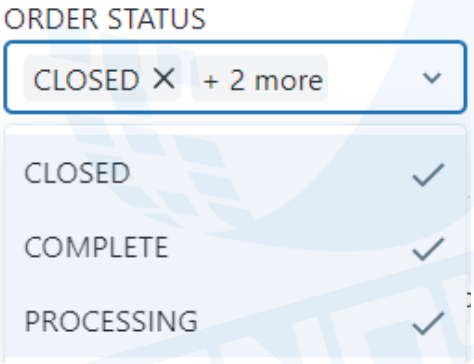
dbutils.widgets.dropdown(name = 'orderstatus' , defaultValue = 'CLOSED' , choices = ['CLOSED', 'COMPLETE' , 'PROCESSING'] , label = 'ORDER STATUS')



- **multiselect**

  //creating the widget multiselect

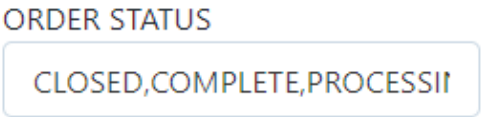
  dbutils.widgets.multiselect(name = 'orderstatus' , defaultValue = 'CLOSED' , choices = ['CLOSED', 'COMPLETE' , 'PROCESSING'] , label = 'ORDER STATUS')



- **text**

  //creating the widget text

  dbutils.widgets.text(name = 'orderstatus' , defaultValue = 'CLOSED' , label = 'ORDER STATUS')



Removing the widgets -

dbutils.widgets.remove('<name-of-widget>')

dbutils.widgets.removeAll()

Passing Parameters from one NoteBook to another

dbutils.notebook.run('<File-Path-of-child-NoteBook>' , <Timeout> , {'<Parameter>' : '<Parameter-Value>'})

Example :

dbutils.notebook.run('/Users/trendytech.sumit@outlook.com/dbutilsdemo' , <Timeout> , {'orderstatus' : 'CLOSED'})

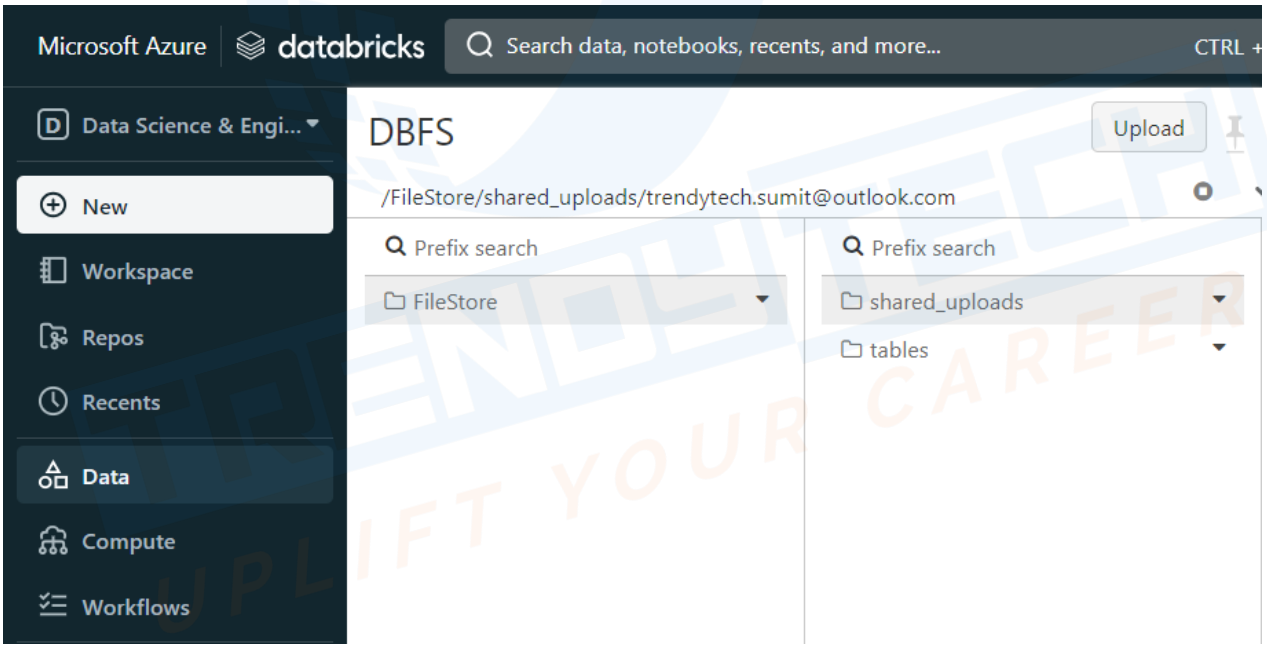
Mount Point

Mount point is used to mount the given source data directory (like the Azure Blob) into Databricks File System (DBFS)

dbutils.fs.help('mount') To get more information about mount points.

Mounting the data present in the Azure Storage Account to DBFS

Step 1 - Create a Storage account -> Create Container -> Upload the data files.



As seen in the above image, no data files are present in the DBFS.

- In order to make the data files uploaded to the Azure Storage Account Container to be available in DBFS, it is required to **mount** the data to DBFS.

  dbutils.fs.mount(source, mount_point, extra_configs)

source

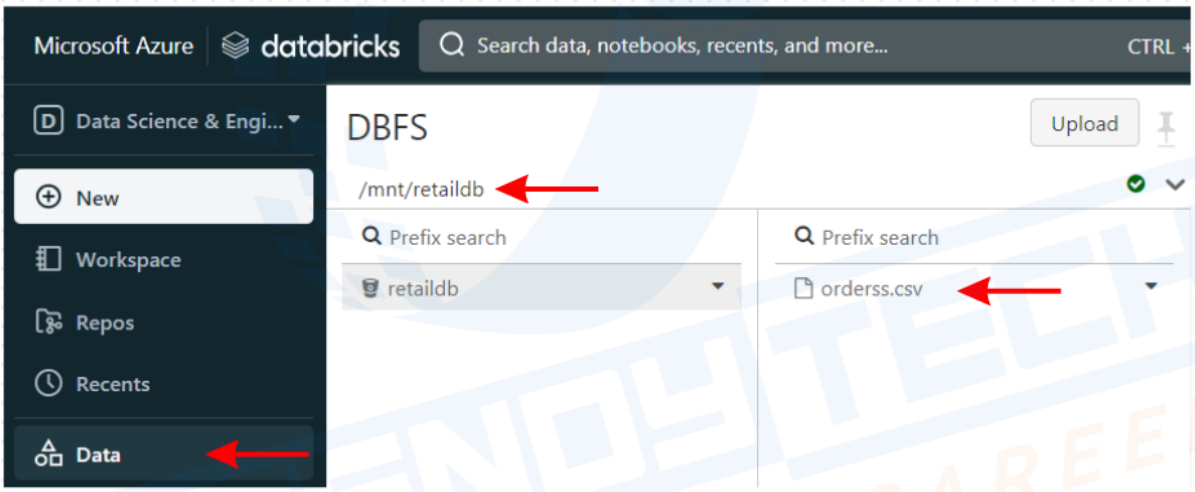
wasbs://<Container-Name>@<Storage-Account-Name>.blob.core.windows.net

mount_point Could be any name for the mount point

extra_configs {'fs.azure.account.key.<Storage-Account-Name>' : '<Storage-Account-Access-Key-Value>'}

Example

```
dbutils.fs.mount(source
='wasbs://inputdatasets@ttstorageaccount100.blob.core.windows.net',
mount_point = '/mnt/retaildb', extra_configs =
{'fs.azure.account.key.ttstorageaccount100.blob.core.windows.net' :
'/kvle+XzSEiOTKGEVHXKpX2E48NE0LL61AkP2rkmM0i9FWB6o4xbdojb/1m
YqhIGB4BsboV3KyQG+AStbRyF5Q=='}
```



Data is now mounted and is present in the DBFS file system

Note : Actual data is not present in DBFS but in the Azure storage. However, operations can be performed on this mounted data as if it were local to DBFS.

Updating and Unmounting the Mount Point

dbutils.fs.mounts() Displays all the mount points

dbutils.fs.unmount('<mount-point>') Removing the mount point

dbutils.fs.updateMount('<old-mount-point-name>' , '<new-mount-point-name>' , True) Updating the name of mount point

Databricks CLI Setup

Step 1: Download and Install Python 3

Step 2: Ensure pip is installed

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
```

Step 3: Install Databricks CLI using

```
pip install databricks-cli
```

Step 4: Configure to connect to the databrick workspace

```
databricks configure --token
```

(For Token, in Databricks Workspace Account -> Settings -> User Setting -> generate token)