**Databricks**

**Cluster**

A diagram of a company

Description automatically generatedThe driver node orchestrates tasks and assigns them to worker nodes. Clusters allow us to create these as a single compute engine, via the driver node. Databricks clusters allow us to run different types of workloads such as ETL!

There are two types of cluster:

Al Purpose

Created manually, via the GUI or the CLI or the API.

* Persistent
* Suitable for interactive workloads
* Shared among many users
* Expensive to run

Job Cluster

Created when a job starts to executed and said job has been configured to use the job cluster.

* Terminated at the end of the job
* Suitable for automated workloads; ETL pipeline.
* Isolated just for the job being executed.
* Cheap to run.

On the compute tab, we can see the different types of computes:

A screenshot of a computer

Description automatically generated

**Cluster Pools**

Normally it may take some time for a cluster to spin up. If you do not want to have to wait, you can have a pool of resources waiting for you via cluster pools.

**Cluster Configuration**

A screenshot of a computer

Description automatically generatedSingle/Multi Node

Multi Node:

* Allows us to configure multiple workers to aid in computation. There is a driver node and multiple worker nodes.

Single Node:

* Has only one node; the driver node and there are no worker nodes.

Access Mode:

A screenshot of a computer

Description automatically generated

Single User: Only one user access. Supports Python, SQL, Scala, R.

Shared: Multiple User Access, only available in Premium. Supports Python, SQL. Provides process isolation, which means that if one process fails, it does not affect the others.

No Isolation Shared: Also supports multiple user access. It’s available in Standard and premium work spaces. Supports Python, SQL, Scala and R. The main difference is that it does not provide process isolation. Failure in one process may affect the others!

A screenshot of a computer

Description automatically generatedDatabricks Runtimes:

The set of core libraries that runs on Databricks.

Databricks Runtime

* Spark
* Scala, Java, Python, R
* Ubuntu Libraries
* GPU Libraries
* Delta Lake

Databricks ML

* Everything from Databricks Runtime…Plus:
* Popular ML Libraries such as:
* PyTorch, Keras, TensorFlow, XGBoost.

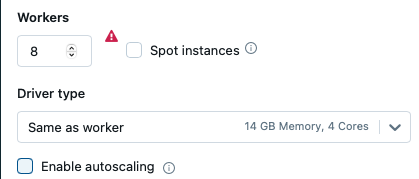
Auto Termination

A screenshot of a computer

Description automatically generated

Terminates the cluster after ‘x’ minutes of inactivity. Default value is 120 minutes. Users can specify a value between 10 and 10000 mins as the duration.

AutoScaling

A screenshot of a phone

Description automatically generatedLeft ON. Right OFF.

Allows the user to specify the minimum and maximum amount of worker nodes. It will auto scale between min and max based on the workload. Not recommended for streaming workloads.

Cluster Types

A screenshot of a computer

Description automatically generated**Memory Optimised**

* Optimised for memory intensive operations.
* For example a Machine Learning workload that caches a lot of data in memory.

**Compute Optimised**

* Optimised to handle workloads that require high computational power, rather than high I/O or memory capacity.
* More cost efficient for CPU intensive tasks.
* Structured streaming operations where you need to make sure that the processing rate is above the input rate above peak times in the day.
* Used for distributed analytics and data science applications
* Machine Learning: training complex learning models, especially those that require significant computational power.
* Data Processing – performing large-scale data transformations and aggregations.

**Storage Optimised**

* Where processes require high disk Input/Output Performance. Often equipped with SSD’s.
* Typically optimised for data processing. They are able to handle large-scale ETL processes, data warehousing and big data analytics where data read/write speeds are crucial for performance.

**General Purpose**

* Balanced Resources: offering a good balance of CPU, memory and storage, making them versatile for different types of workloads.
* They are flexible: suitable for diverse applications including data processing, analytics and machine learning.
* They are cost Efficient: designed to provide adequate performance for a wide range of tasks without being overly specialised.
* Typical Use Cases:
  + Data Exploration: running ad-hoc queries and exploratory data analysis.
  + ETL Workloads: Performing extract, transform and load operations on medium sized datasets.
  + Machine Learning: training and deploying machine learning models that do not require extreme computational power or memory.
  + Business Intelligence: Supporting BI tools and dashboards for reporting and analytics.

**GPU Accelerated**

* High Computational Power: GPUS provide significant computation power, especially for parallelisable tasks.
* Efficient for deep learning, neural network training and other tasks that benefit from parallel computation.
* Utilise GPU instances like NVIDIA Tesla, which are optimised for compute-heavy operations.
* Typical Use Cases:
  + Deep Learning: training large-scale neural networks and deep learning models.
  + Machine Learning: accelerating the training of machine learning models that require extensive computation.
  + High-Performance Computing (HPC): Applications in scientific computing, simulations and engineering that benefit from GPU acceleration.
  + Data Processing: Specific ETL tasks that can leverage GPU for faster processing.

Cluster Policy

Administrators are able to create custom cluster policies. They can assign them to users or groups.

A screenshot of a computer

Description automatically generatedHere there are far fewer options. The runtime version has been set, the node type has been set and the auto termination has already been set.

This has been set to a Machine Learning (ML) cluster type, so could be useful for machine learning engineers.

* Simplifies the user interface.
* Enables standard users to create clusters.
* Achieves cost control by prohibiting more expensive computational resource assignment.

**Creating a Cluster**

The LTS suffix stands for Long-term Support. Typically these runtime versions are supported for 2 years.

A screenshot of a computer

Description automatically generated



Photon acceleration can help speed up queries. It is more useful for larger queries as this can be used to complete them more quickly. However, for smaller workloads, it is not useful.

Node type standard:

A close-up of a computer screen

Description automatically generated

Set auto-terminate:

A screenshot of a computer

Description automatically generated

**Pricing**

Price can depend on several factors.

* Workload (All purpose/jobs/sql/photon)
* Tier (Premium/Standard)
* VM Type (General Purpose/GPU/Optimised)
* Purchase Plan (Pay as you Go / Pre-Purchase)

A screenshot of a computer

Description automatically generated

Here we can see the summary for our cluster. A Databricks Unit (DBU) is a normalised unit of processing power on the Databricks Lakehouse Platform used for measurement and pricing purposes.

We also need to pay for the virtual machines themselves. We pay for the amount of processing and the virtual machines.

A screenshot of a computer screen

Description automatically generatedHere is the total cost for the cluster:

Here is an example:

A screenshot of a computer

Description automatically generated

To calculate costs it is possible to look up:

DBU Cost:

A screenshot of a computer

Description automatically generatedVM Cost:

A screenshot of a computer screen

Description automatically generatedA screenshot of a phone

Description automatically generatedHere is the calculation. Note that there are no workers hence the worker cost will be 0:

**Azure Pricing Portal**

Fortunately Azure has a pricing portal that allows you to pre-calculate costs:

<https://azure.microsoft.com/en-gb/pricing/details/databricks/>

A screenshot of a computer

Description automatically generated

**Cost Control**

From the azure portal. Cost management allows us to prevent excess costs.

A screenshot of a web page

Description automatically generated

To set an alert we can go to ‘cost alerts’. We can specify custom monthly expenditures.

**Cluster Pool**

Creating a cluster often takes time for the cluster to spin up. In order to minimise the setup time we can use a cluster pool. The key features of a cluster pool:

* Reduced Startup Time: Instances are available so the cluster can be up and running much faster compared to creating a cluster from scratch.
* Cost Efficiency: pools help in reusing instances across multiple clusters, reducing the overall cost. They help in managing idle instances efficiently.
* Scaling: pools can scale the number of instances up and down based on the needs of the workloads. This ensures that resources area available when needed but not wasted when they are not.

**Cluster Policy**

Cluster policies in Databricks are a way to manage and control the configurations and behaviours of clusters within an organisation. They provide a governance mechanism to enforce best practices, security and cost controls by defining a set of rules and constraints for cluster creation and management.

An administrator can define a ‘standard policy’. In it, he can hide attributes, fix values or set default values. Here is an example in JSON format:

{

"name": "Standard Policy",

"definition": {

"spark\_version": {

"type": "fixed",

"value": "auto:latest-lts"

},

"node\_type\_id": {

"type": "enum",

"values": ["m5.large", "m5.xlarge"]

},

"autoscale": {

"type": "range",

"min": 2,

"max": 10

},

"autotermination\_minutes": {

"type": "range",

"min": 10,

"max": 60,

"defaultValue": 30

},

"enable\_elastic\_disk": {

"type": "fixed",

"value": true

}

}

}

A blue icon with white text

Description automatically generatedNow, when the user comes to choose a policy, pre-defined options will have already been selected.

**Databricks Notebooks**

A notebook is a series of cells that can run commands.A screenshot of a computer

Description automatically generated

To run the notebook we can select ‘run all’. This will run all of the code in the cells.

The cluster needs to be attached to a cluster, so that computational power is allocated to it! To do this, you can select the cluster you want to use from the drop down menu.

A screenshot of a computer

Description automatically generated

The ‘share’ button allows you to share the A screenshot of a computer

Description automatically generatednotebook with the rest of the team.

By using the cells, it allows us to manage how our code is set out.

Note that if you try to run one cell before running the others, the variables; message\_one and message\_two, will not have been defined, therefore you get an error message. In order to print correctly you need to ‘Run all’.

A screenshot of a computer

Description automatically generated

**Magic Commands**

Magic commands allow Databricks to switch over to different syntax. For example, using ‘**%sql’** allows us to write in SQL syntax.

In this instance the context of the cell is set to SQL.

It is also possible to change the context to scala; ‘**%scala’**.

**‘%md’**

Is the mark down language which allows us to comment our code. A screenshot of a computer

Description automatically generated

‘**%fs’**

File system command. Allows you to run file system commands such as ‘ls’ and ‘cd’.

**‘%sh’**

The shell command can be used to see all the processes that are running.

**Databricks Utilities**

dbutils

‘dbutils is a utility provided in Databricks to facilitate various tasks, such as working with files, managing secrets and handling data.

‘dbutils.fs’ allows you to interact with the Databricks file system and other file systems. Functions may include commands to list, copy, move, remove files and more.

‘dbutils.secrets’ manages secrets such as passwords, tokens, and other sensitive information. Enables secure storage and access to these secrets within your notebooks.

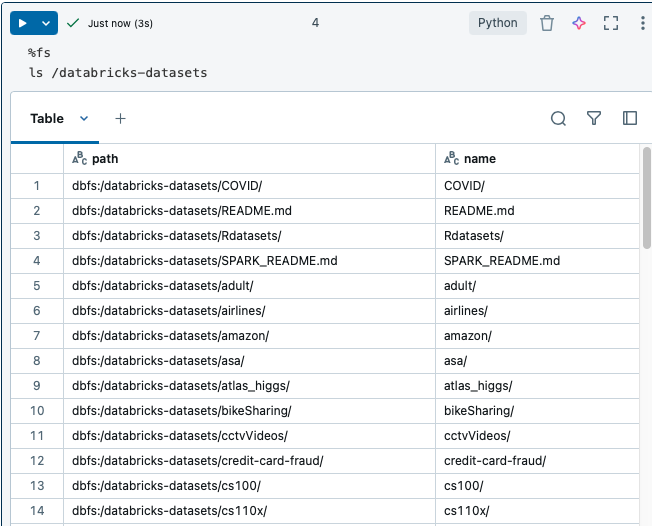
‘dbutils.notebook’ allows you to execute other notebooks from within a notebook.

‘dbutils.widgets’ allows the creation and handling of input widgets for parameterising notebooks. Supports different types of widgets, such as text boxes, dropdowns and multi-selects.

‘dbutils.library’ manages libraries within a notebook session. Functions include commands to install, list and restart libraries.

‘dbutils.utilities’ provides miscellaneous utility functions.

Here we can see that all the files in the ‘databricks-datasets’ folder have been listed:



A screenshot of a computer

Description automatically generatedWe can use the dbutils.fs.ls(‘…’). In this output, we are outputting exactly the same information. However, the output of this, is in the form of a Python list. This means that we can iterate through the list and do things with the data!

By using dbutils we can get greater flexibility as we can now use other languages such as python to interact with the data

A screenshot of a computer program

Description automatically generatedHere we can combine dbutils with python. We can iterate through the list of files and folders. We can then print all of the folders that exist.

Dbutils.help()

Lists the available utilitiesA screenshot of a computer

Description automatically generated

For a more detailed look at individual utilities:

Dbutils.fs.help()

A screenshot of a computer

Description automatically generated

We can also pass in an argument:

Dbutils.fs.help(‘ls’)A screenshot of a computer

Description automatically generated

**DBC Archive**

**A screenshot of a computer

Description automatically generated**Allows us to backup one folder so that we can make regular backups of our project.

This DBC file can be exported into another workspace so that you can begin working on the files again.

**Access Azure Data Lake**

A screenshot of a computer diagram

Description automatically generated

How can we connect Databricks to Azure data lake storage gen2?

**Connecting Via Access Key**

Anatomy of the URL:

We can see that in one of our containers, it is named ‘demo’. The storage account name is; f1dl9072024

A screenshot of a computer

Description automatically generated

dbutils.fs.ls("abfss:://demo@f1dl9072024.dfs.core.windows.net")

This is the url. However, we now need an access key. A screenshot of a computer

Description automatically generated

spark.conf.set("fs.azure.account.key.f1dl9072024.dfs.core.windows.net",

"qH8uTGYGhICJUtf6VV3D3jNPjMpHzV80MzQDfvBN9Z+FyPSfyIRYkfIN6hIEU1+UBl73UEH2dXm6+AStt3oC5g==")

A screenshot of a computer

Description automatically generatedThis is our spark configuration. So now our notebook should look like this:

And we can see that it has correctly listed files stored in our container.

**Read the file**

spark.read.csv("abfss://demo@f1dl9072024.dfs.core.windows.net/upload-circuits-csv")

A screenshot of a computer

Description automatically generatedWe can read the file which will return a dataframe:

A screenshot of a computer

Description automatically generatedTo see the data we can use the display function:

So far we have looked at access keys. However, another method of providing access is with a ‘Shared Access Signature’; ‘SAS Token.’

**Accessing via Shared Access Signature (SAS)**

* Provides fine grained access to the storage
* Restricts access to specific resource types/services
* Allows specific permissions
* Restrict access to specific time period
* Limit access to specific IP addresses
* Recommended access pattern for external client

SaS tokens are useful for external clients who should not be trusted with full access to storage.

**Configuring Notebook for SAS Token**

Here is the documentation from Microsoft: <https://learn.microsoft.com/en-us/azure/databricks/connect/storage/azure-storage>

In it we can see the code necessary for our purposes.

A screenshot of a computer

Description automatically generated

Now that we have the correct code, we can implement it into our notebook.

**Generate SAS Token**

A screenshot of a computer

Description automatically generated

Generate the sas token by going to the storage container. It is possible to select an individual folder!

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedNow we can edit our code accordingly;

And it has worked!

A screenshot of a computer program

Description automatically generated

**Connecting to KeyVault**

**Databricks Secret Scope**

A Databricks secret scope is a secure storage mechanism within Databricks for managing sensitive information, such as API keys, passwords, and other credentials. By using secret scopes, you can store secrets securely and access them in notebooks and jobs without exposing them directly in the code. Secrets are encrypted and managed securely to prevent unauthorised access.

**Secret Scope Menu**

There is a secret menu for the creation of secret scopes. From the url at the top of the web page simply add; ‘#secrets/createScope’.

The complete url will look something like this;

https://adb-2110174253848098.18.azuredatabricks.net/?o=2110174253848098&l=en#secrets/createScope

A screenshot of a computer

Description automatically generated

Now that the secretscope has been created, it should be possible for databricks to access stored secrets!

To find the DNS and Resource ID. Go to the Key Vault and select ‘properties’.

A screenshot of a computer

Description automatically generated

The Vault URI and resource ID are displayed.

A screenshot of a computer

Description automatically generated

Click create and the scope has been created.

**Accessing Secret**

# Access secret:

secret = dbutils.secrets.get(*scope*="f1-key-vault-scope", *key*="storage-account-primary-key")

To access the secret we need to use ‘dbutils.secrets.get()’. The scope is the name of the vault repository; in this case it is f1-key-vault-scope. The key is the name of the secret that you need. ‘Secret’ should now evaluate to the secret named ‘storage-account-primary-key’ which is stored in the ‘f1-key-vault-scope’ vault.

First we access the secret.

A screen shot of a computer code

Description automatically generatedThen we set the spark configuration.

A screenshot of a computer

Description automatically generatedNext we can verify that our circuits.csv file is in our demo folder. To do this we use the dbutils.fs.ls() command.

A screenshot of a computer

Description automatically generatedFinally we can view the data stored in our csv file:

**Access ADL using Service Principal**

Service Principals re similar to user accounts. A diagram of a program

Description automatically generated

This is the recommended method of connection for CI/CD pipelines or databricks jobs.

They provide better security and traceability. The steps are as follows:

1. Register Azure AD Application / Service Principal.
2. Generate a secret password for the application.
3. Set spark config with App client ID, Directory Tenant ID and secret
4. Assign Role ‘Storage Blob Data Contirbutor’ to the Data Lake.

Register Azure Active Directory

From the Azure portal go to; ‘Microsoft Entra ID’.

A screenshot of a computer

Description automatically generated

Under manage select ‘App registrations’ and then select ‘New registration’.

A screenshot of a computer

Description automatically generatedClick register and now we have created a service principal:

The application has been successfully created.

A screenshot of a computer

Description automatically generated

We will need the Application (client) ID and the Directory (tenant) ID.

Generate a secret/password

A screenshot of a white background

Description automatically generatedNext we need to go to Certificates & Secrets which is also under manage.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedClick add and we have now created the secret.

A white board with red numbers and letters

Description automatically generatedMake sure to copy the secret ‘Value’. You can now save them to variables;

Configure Spark Configuration

Microsoft documentation: <https://learn.microsoft.com/en-us/azure/databricks/connect/storage/azure-storage>

A screenshot of a computer

Description automatically generated

Final spark configuration;

spark.conf.set(f"fs.azure.account.auth.type.{storage\_account}.dfs.core.windows.net", "OAuth")

spark.conf.set(f"fs.azure.account.oauth.provider.type.{storage\_account}.dfs.core.windows.net", "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")

spark.conf.set(f"fs.azure.account.oauth2.client.id.{storage\_account}.dfs.core.windows.net", f"{client\_id}")

spark.conf.set(f"fs.azure.account.oauth2.client.secret.{storage\_account}.dfs.core.windows.net", client\_secret)

spark.conf.set(f"fs.azure.account.oauth2.client.endpoint.{storage\_account}.dfs.core.windows.net", f"https://login.microsoftonline.com/{tenant\_id}/oauth2/token")

Assign Storage Blob Data Contributor Role to Service Principle

A screenshot of a computer

Description automatically generatedGo to the storage resource and select IAM:

Select; ‘Storage Blob Data Contributor’.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedOn the next page search for the service principal:

A screenshot of a computer

Description automatically generated

Finally click review + assign.

The service principle should now be able to access the storage account.A screenshot of a computer

Description automatically generated

**Terraform Code**

# This will register a new app registration in Microsoft Entra ID:

# Dynamically gain tenant\_id key:

data "azurerm\_client\_config" "current" {}

# Create provider:

provider "azuread" {

tenant\_id = data.azurerm\_client\_config.current.tenant\_id

}

# Creates Azure Active Directory Application (User):

resource "azuread\_application" "setup" {

display\_name = "formula1-app"

}

# Create Service Principal for the Application

resource "azuread\_service\_principal" "setup" {

client\_id = azuread\_application.setup.client\_id

}

# Create a Client Secret for the Application

resource "azuread\_application\_password" "setup" {

application\_id = azuread\_application.setup.id

display\_name = "f1-secret"

end\_date\_relative = "8760h" # 1 year

depends\_on = [ azuread\_application.setup ]

}

# Assign Storage Blob Data Contributor Role to Service Principal

resource "azurerm\_role\_assignment" "blob\_data\_contributor" {

principal\_id = azuread\_service\_principal.setup.object\_id

role\_definition\_name = "Storage Blob Data Contributor"

scope = azurerm\_storage\_account.storage\_account\_one.id

}

Finally upload data to keyvault:

# Store App User Information:

resource "azurerm\_key\_vault\_secret" "app\_client\_id" {

name = "application-client-id-demo"

value = azuread\_application.setup.client\_id

key\_vault\_id = azurerm\_key\_vault.f1keyvault.id

depends\_on = [ azuread\_application.setup ]

}

resource "azurerm\_key\_vault\_secret" "tenant\_id" {

name = "directory-tenant-id-demo"

value = data.azurerm\_client\_config.current.tenant\_id

key\_vault\_id = azurerm\_key\_vault.f1keyvault.id

depends\_on = [ azuread\_application.setup ]

}

resource "azurerm\_key\_vault\_secret" "client\_secret" {

name = "application-client-secret"

value = azuread\_application\_password.setup.value

key\_vault\_id = azurerm\_key\_vault.f1keyvault.id

depends\_on = [ azuread\_application.setup ]

}

**Session Scoped Authentication**

Session scoped authentication refers to a method of managing user authentication that is tied to a specific session.

**Cluster Scoped Authentication**

Another method of providing authentication, is by authenticating the cluster. This way, any notebook that runs on that cluster will have access to the corresponding Data Lake.

The limitations of this are; if data engineers need access to a data lake, but data analysts should not have access, then two separate clusters will need to be created. It is therefore not a widely used approach.

A screenshot of a computer

Description automatically generatedEdit the compute configuration it should contain the following:

It should contain this path: fs.azure.account.key.f1dl9072024.dfs.core.windows.net

And the secret access key:

9u29xfJs7jCKNibtmmPrOcGG00MmhmacFAve/qxiXE3i3gY2qml3yqSYGuzIGgI1iwGhJZiSbksu+ASta4+FuA==

Now when we go back to our notebook we don’t have to add our spark conf information!

A screenshot of a computer

Description automatically generated

Terraform Code:

resource "databricks\_cluster" "scoped\_cluster" {

cluster\_name = "f1-cluster-scoped-authentication"

node\_type\_id = data.databricks\_node\_type.smallest.id

spark\_version = data.databricks\_spark\_version.latest\_lts.id

autotermination\_minutes = 25

# CONFIGURATION FOR SINGLE NODE CLUSTER!!!!

num\_workers = 0

spark\_conf = {

# Single-node

*"spark.databricks.cluster.profile"* : "singleNode",

*"spark.master"* : "local[\*]",

"fs.azure.account.key.${azurerm\_storage\_account.storage\_account\_one.name}.dfs.core.windows.net" : azurerm\_storage\_account.storage\_account\_one.primary\_access\_key

}

custom\_tags = {

*"ResourceClass"* = "SingleNode"

}

}

Note that this method accesses the security key during the Terraform configuration.

To access the access key from the key vault:

resource "databricks\_cluster" "scoped\_cluster" {

cluster\_name = "f1-cluster-scoped-authentication"

node\_type\_id = data.databricks\_node\_type.smallest.id

spark\_version = data.databricks\_spark\_version.latest\_lts.id

autotermination\_minutes = 25

# CONFIGURATION FOR SINGLE NODE CLUSTER!!!!

num\_workers = 0

spark\_conf = {

# Single-node

*"spark.databricks.cluster.profile"* : "singleNode",

*"spark.master"* : "local[\*]",

"fs.azure.account.key.${azurerm\_storage\_account.storage\_account\_one.name}.dfs.core.windows.net" : "{{secrets/f1-scope/storage-account-primary-key}}"

}

custom\_tags = {

*"ResourceClass"* = "SingleNode"

}

}

A screenshot of a computer

Description automatically generatedThis produces a configuration that looks like this:

**AAD Credential Passthrough**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedEnable credential passthrough for user-level data access:

Now A screenshot of a computer

Description automatically generatedassign IAM role storage blob data contributor:

Add user:

A screenshot of a computer

Description automatically generated

**Databricks Mounts**

**Databricks File System (DBFS)**

A diagram of a computer

Description automatically generated

DBFS is a distributed file system built on top of Azure Blob Storage and optimised for Databricks. It provides a familiar file system interface that allows users to read and write data as if they were interacting with a local file system, but with the scalability and durability of cloud storage.

**Explore DBFS Root**

A screenshot of a computer

Description automatically generatedFirst, we can list all the folders in the DBFS root with ‘dbutils.fs.ls(‘/’)’ command.A screenshot of a computer

Description automatically generated

In order to upload files, enable DBFS File Browser in settings.

To access the DBFS FileStore click on ‘add data’. A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedThen upload the file:

Now when we run the dbutils.fs command again, we will see the FileStore appear:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedWe can view the file that we have just uploaded: make sure the file path is correct.

We can read the data from the CSV file:

A screenshot of a computer

Description automatically generated**NOTE:** managing access control at the root directory is difficult. It is hard to implement fine-grained access control policies, increasing the risk of unauthorised access or accidental data exposure.

**Databricks Mounts**

Using the dbfs root for keeping sensitive data is not a good idea! It becomes easily accessible. If we should not use DBFS root, how should we store sensitive data?

Advantages of mounts;

1. **Unified data access**: simplified access; mounting external storage allows users to access data stored in Azure Blob Storage, Azure Data Lake Storage, or Amazon S3 using standard file system paths. This makes it easier to work with data as if it were stored locally in DBFS.
2. **Integration with Databricks Workflows**; mounted storage integrates seamlessly with Databricks notebooks, jobs, and clusters, making it straightforward to read from and write to the mounted storage within your data processing workflows.
3. **Security and Access Control**; granular access control allows us to leverage cloud provider-specific security features such as Azure RBAC (role based access control). This enhances security by ensuring that only authorised users can access data.

Some other benefits;

* Access data without requiring credentials.
* Access files using file semantics rather than storage URLS.
* Store files to object storage (e.g. Azure Blob), so you get all the benefits from Azure.

Recommended solution for access Azure Storage until the introduction of Unity Catalogue (Generally available form the end of 2022).

**Mounting Azure Data Lake Storage Gen2**

Documentation: <https://learn.microsoft.com/en-us/azure/databricks/dbfs/mounts>

Note the configuration from the documentation:

A screenshot of a computer code

Description automatically generated

This will allow us to connect.

A screenshot of a computer program

Description automatically generatedHere is the updated code:

A screen shot of a computer code

Description automatically generatedNext update the dictionary:

Then mount the container:

A screenshot of a computer

Description automatically generated

Notice that the output is true which means the file system has been mounted successfully. Now we can use the ‘/mnt/’ nomenclature to connect to our data:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedList the mounts: dbutils.fs.mounts().

This command is useful to list all of the mount points. Some people might not follow naming conventions.

It is possible to unmount with dbutils.fs.unmount(‘/mnt/…./…’)A screenshot of a computer

Description automatically generated

**Mounting Additional Containers**

A screenshot of a survey

Description automatically generatedPreviously we only mounted the container; ‘demo’. But there are others:

Lets create a python function that can help us!

First define the function:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generatedThen call the function: