NO.

Title

Enjoy the most efficient handwriting experience! Taking notes on the go, whether for inspiration, ideas, knowledge learning, business insights, or even sketches...

NOTE IN

Creative notes

By reading we enrich the mind;
by writing we polish it.

- Apache Spark coordinates work across multiple processing nodes in a cluster, known in Microsoft Fabric as Spark pool
- The process of distributing tasks and collating results is handled by spark

Spark Pools

- Spark pools consists of compute nodes that distribute data processing tasks. It contains 2 kinds of node:

(1) head node → coordinates distributed process through driver program

(2) multiple worker nodes → on which actual executer perform the actual data processing tasks.

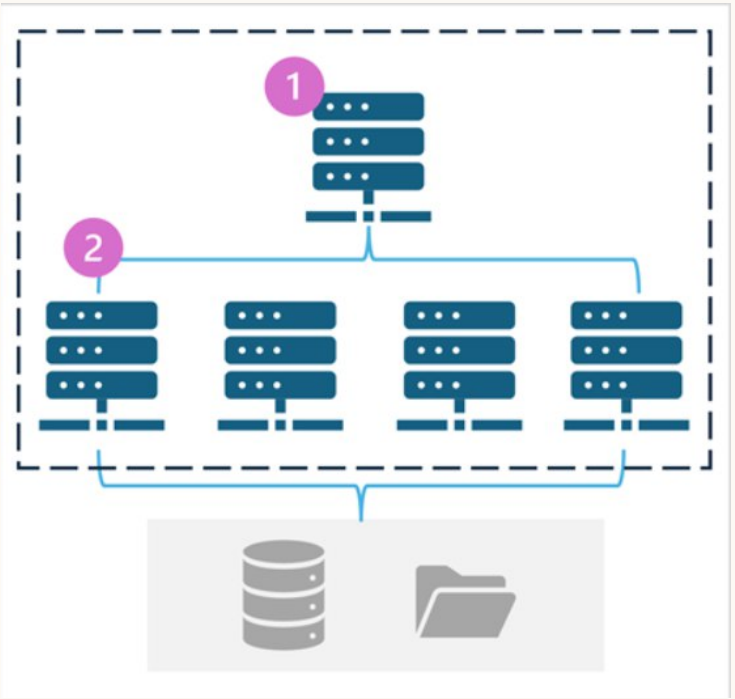
types of Spark Pools

Microsof fabric provides:

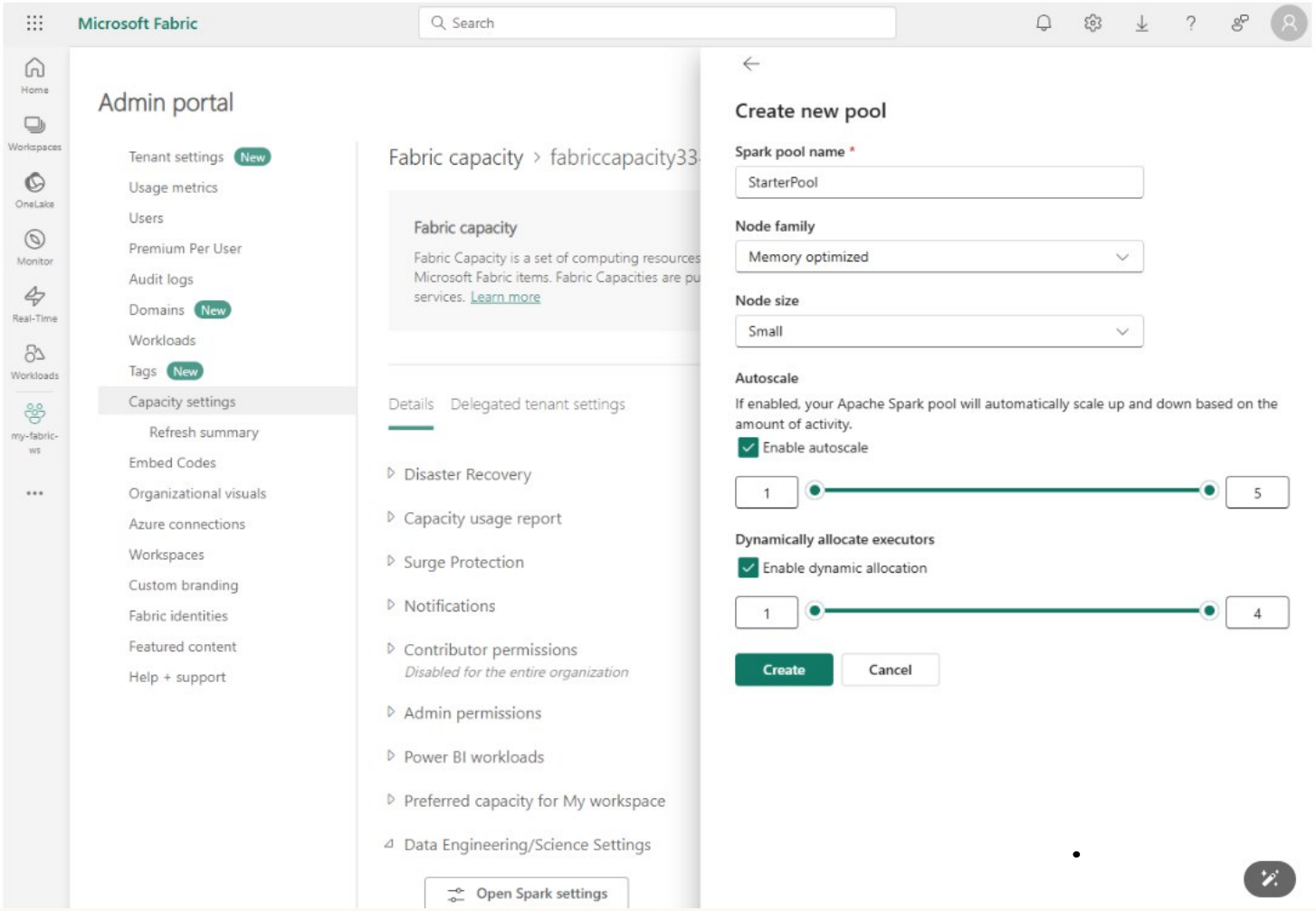
(1) starter pool in each workspace enabling spark jobs to be started & run quickly with minimal setup and configuration (prehydrated compute resource)

(2) custom pool by your own customization of specific node configurations (provided on demand)

Note The ability to customize spark pool settings can be disabled by Fabric administrator at the fabric capacity level



You can manage settings for the starter pool and create new Spark pools in the **Admin portal** section of the workspace settings, under **Capacity settings**, then **Data Engineering/Science Settings.**



- Each workspace is assigned a spark cluster
- workspace admin can manage settings for the spark cluster in the workspace settings

- Configuration settings of spark pools :

Node Family : Type of virtual machines used for spark cluster nodes (e.g. memory optimized nodes)
↳ dependent on subscription)

Autoscale : provide nodes as needed (scale up/down) automatically. Initial & max nodes allocated to the pool can be set

Dynamic allocation : allocate executor process on the worker nodes dynamic based on the data volumes

Native execution engine : in microsoft Fabric is a vectorized processing engine that run spark operations directly on lakehouse infrastructure

Runtime Version

It improve performance of queries used on large datasets in parquet or Delta fil formats

To enable the native execution engine for a specific script or notebook, you can set these configuration properties at the beginning of your code, like this:

JSON Copy

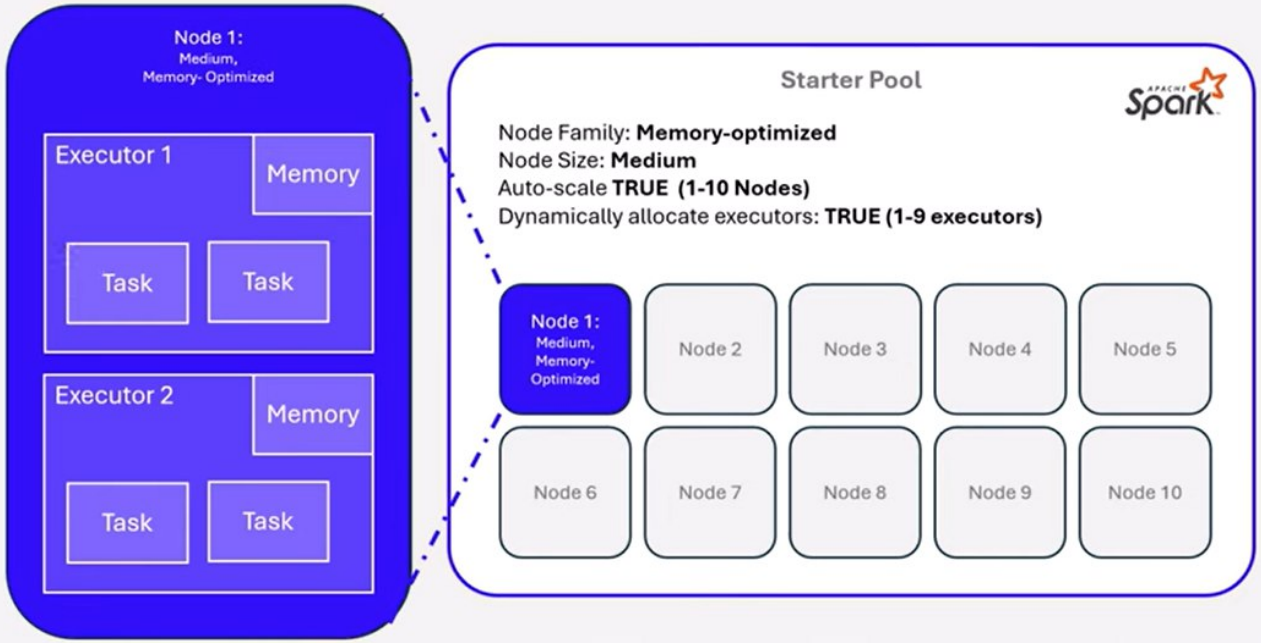
```
%%configure
{
   "conf": {
       "spark.native.enabled": "true",
       "spark.shuffle.manager": "org.apache.spark.shuffle.sort.ColumnarShuffleManager"
   }
}
```

Spark properties

High concurrency mode : share the spark session across multiple concurrent users or process

(1) Isolation between notebooks
(2) same session so "increase efficiency

A Node is a commodity server. Executors are JVM processes that performs Tasks.

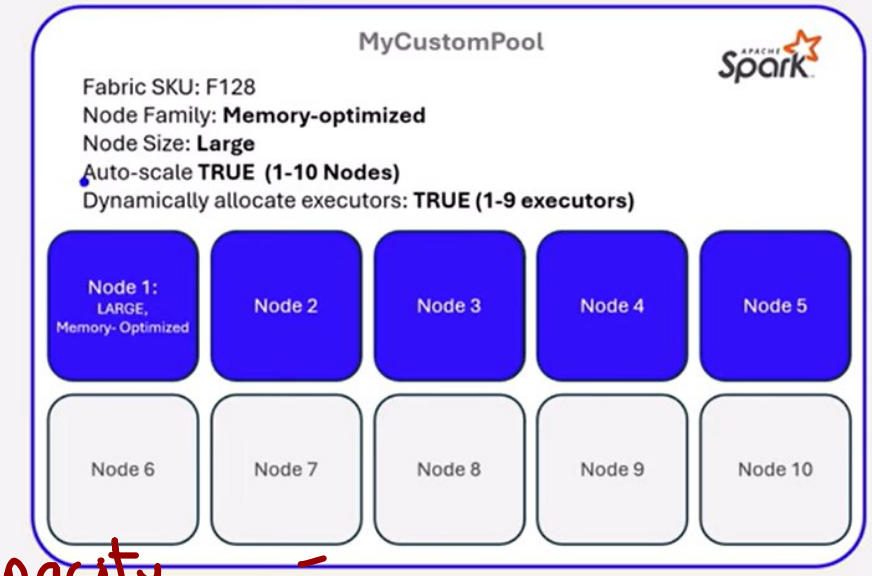


So instead of a Starter Pool, you want to create a Custom Spark Pool

Beware:

Starter Pool
Spark Session startup time:
~10 seconds

Custom Spark Pool –
Spark Session startup times:
~2-3 minutes



Why we create it?

- Consume less Capacity

SKU name	Capacity units	Spark VCores	Node size	Default max nodes	Max number of nodes
F2	2	4	Medium	1	1
F4	4	8	Medium	1	1
F8	8	16	Medium	2	2
F16	16	32	Medium	3	4
F32	32	64	Medium	8	8
F64	64	128	Medium	10	16
(Trial Capacity)	64	128	Medium	10	16
F128	128	256	Medium	10	32
F256	256	512	Medium	10	64
F512	512	1024	Medium	10	128
F1024	1024	2048	Medium	10	200
F2048	2048	4096	Medium	10	200

- Data requires more Compute
- Dividing processing to different pools so, They won't eating The same resource when They run

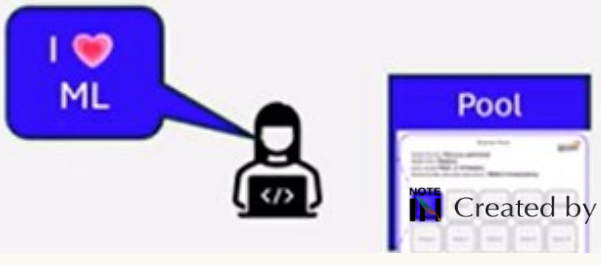
The limits of the Starter Pool, and when you might need to create a Custom Pool

- **Data size**: if you are processing very large datasets, that the Starter Pool is struggling to process, then a custom Pool with increased node sizing can help.

- **Concurrency:** Spark is using a limited set of resources, based on your Fabric Capacity, and if you have multiple users all spinning up their own Spark Sessions, this will eat up your resources. Therefore, when you have multiple users all trying to use Spark (or multiple jobs all scheduled to use the capacity at the same time), you might need a custom Spark configuration to accommodate this.

- **Variance in job resource requirements:** When you have a high variance in the resources required for certain jobs that you are running, it can be more efficient to create custom configurations for each job. For example, some very large jobs, and some very small jobs.

- **Job Type**: our focus currently is on Data Engineering, but Data Scientists/ AI Engineers might also be using your capacity. Training machine learning models will nearly always require a customized Spark configuration.

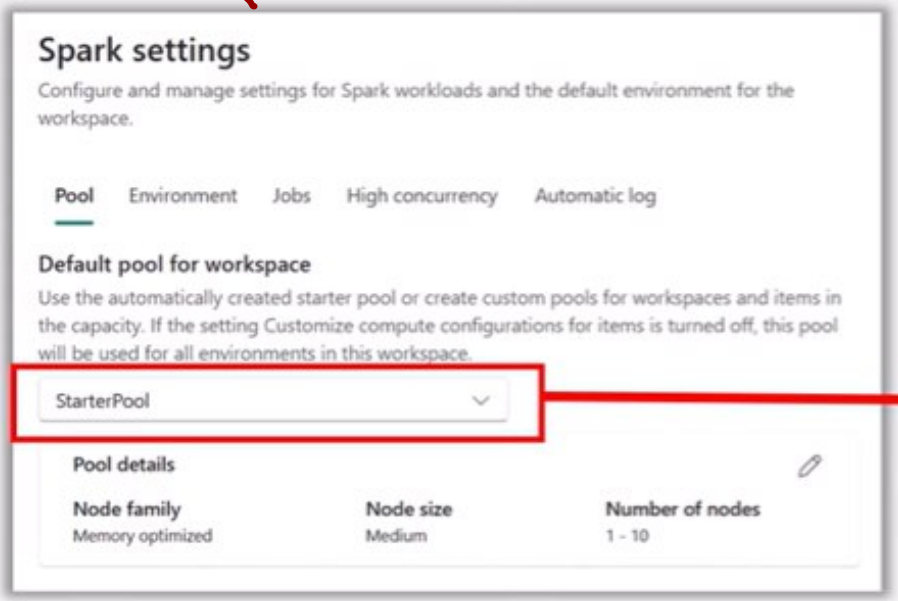


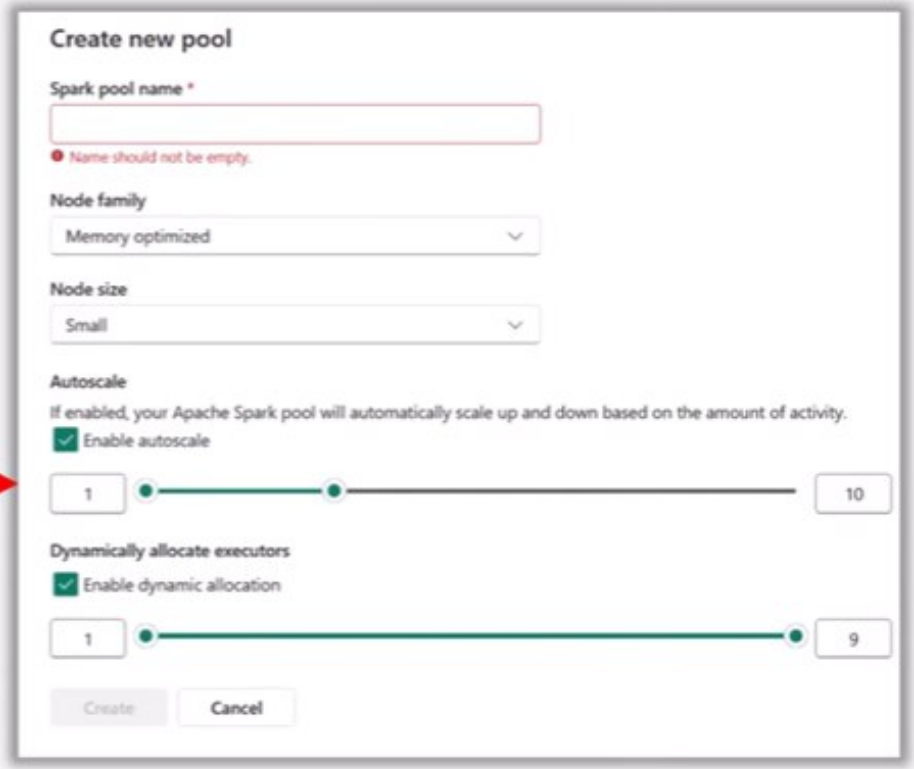
Down Side

Doesn't support CI/CD and is tied to the workspace

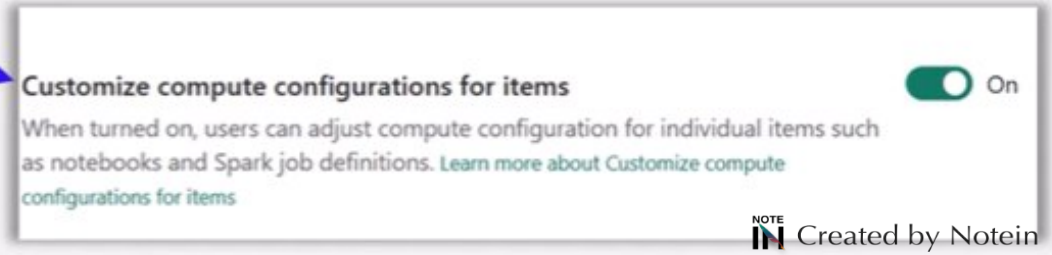
So where can I create and manage custom Spark Pools?

Option 1: In the Workspace Settings





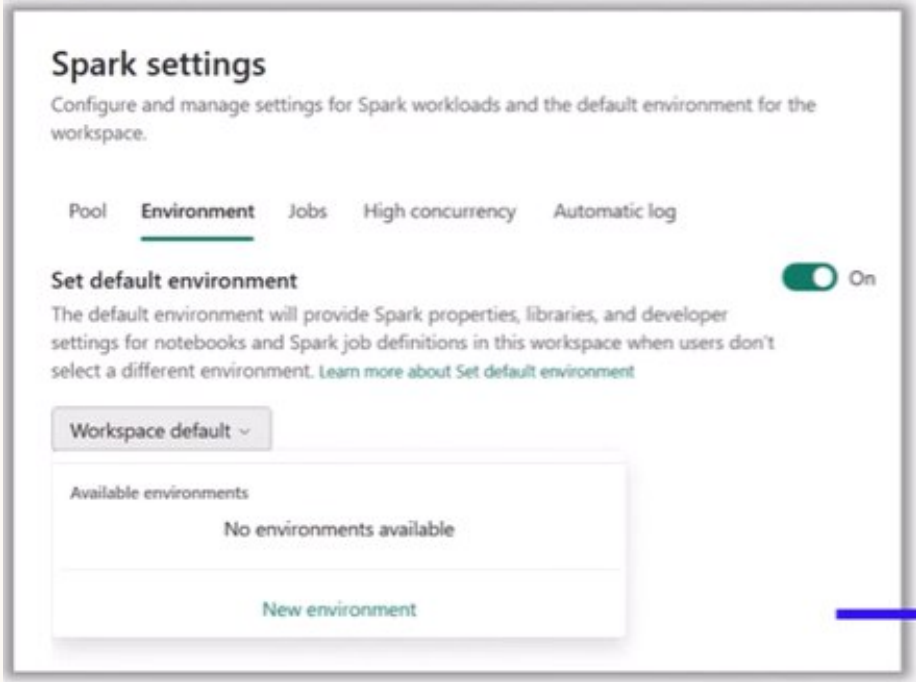
All notebooks and Spark Jobs in the Workspace will use these Settings (by default)

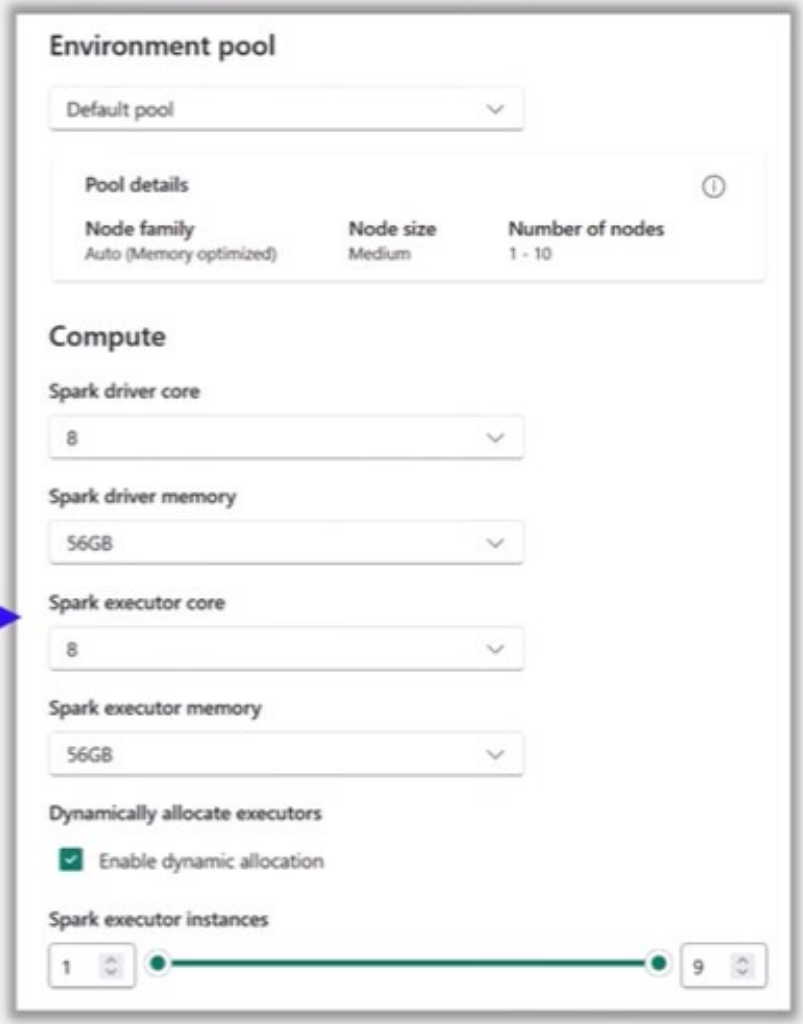


- ✗ *No git integration*
- ✗ *No pool re-use across workspace*

So where can I create and manage custom Spark Pools?

Option 2: a Default Environment in your Workspace Settings





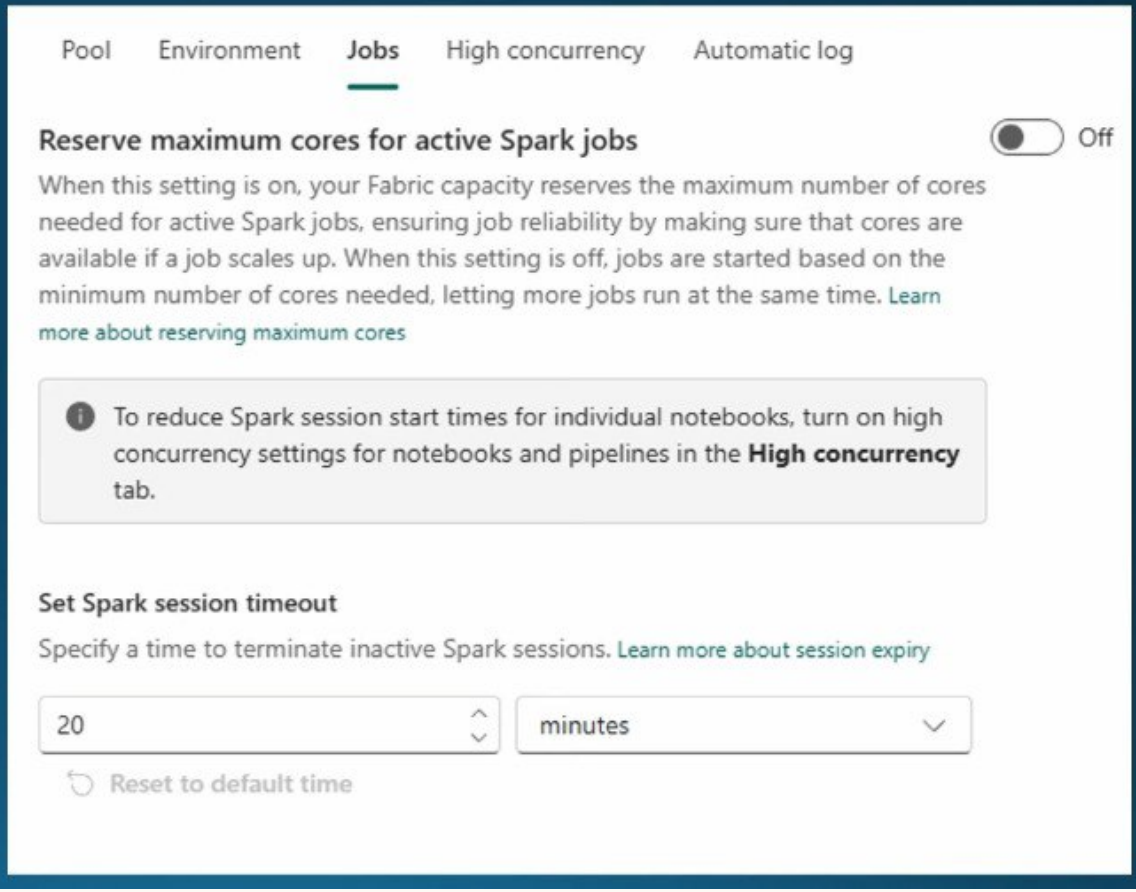
All notebooks and Spark Jobs in the Workspace will use these Settings (by default)

✔ *Allows Git integration (through Environment)*
✘ *No pool re-use across workspace*

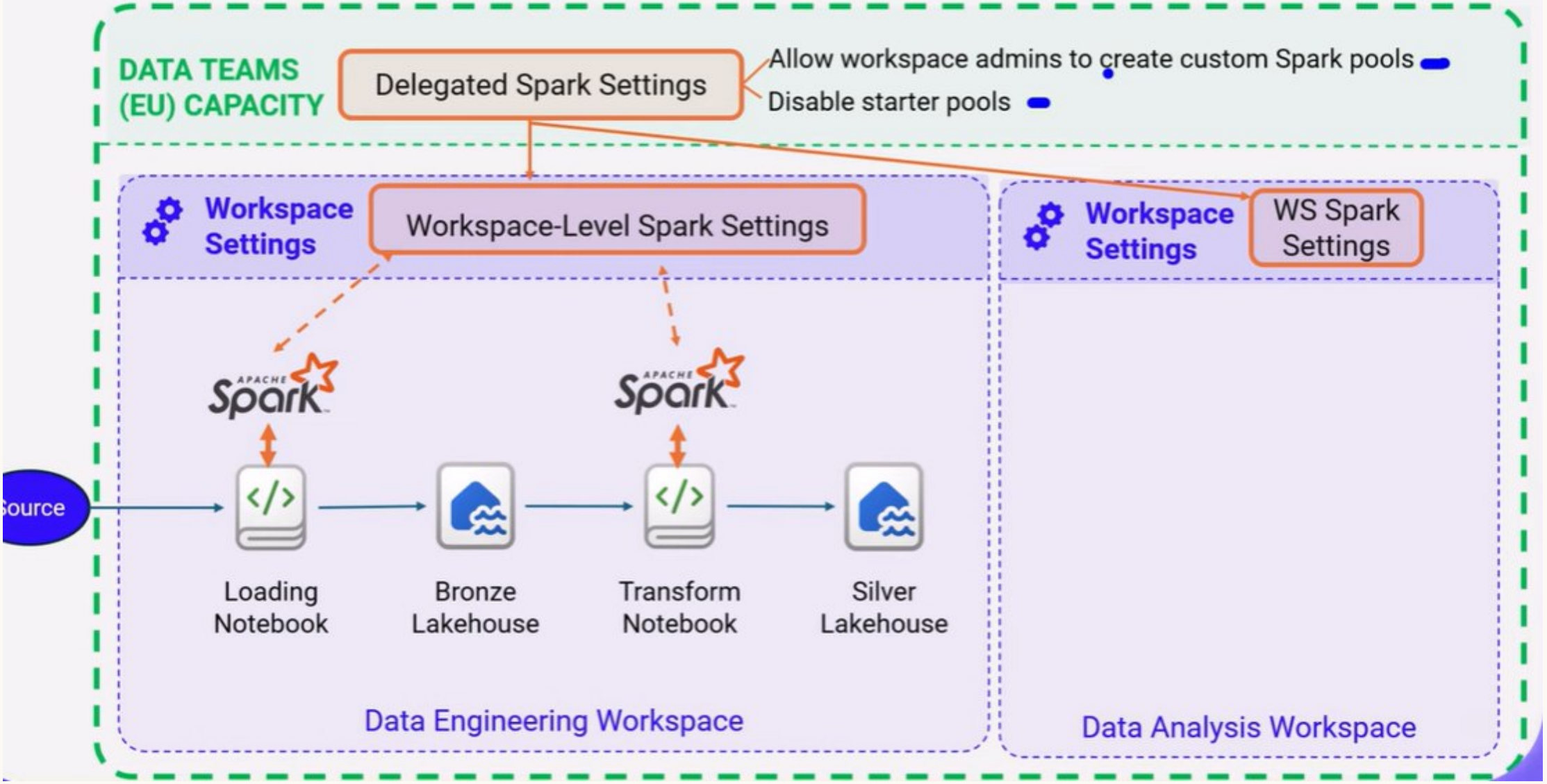
- One workspace can hold multiple environment (one is default env.)

Spark Job Settings

- Reserve maximum cores for active Spark jobs
  - Spark estimates the minimum and maximum number of cores it needs to complete a job and then does the reservations based on that
  - Turning on gives more reliability but could be less optimal usage
  - Turning off reduces reliability but uses the compute more optimally
- Spark session timeout
  - Time after inactive Spark sessions are terminated
  - Defaults to 20 minutes



Configuring Spark Settings – in the Capacity Admin Settings



- Due to massive growth in data, organizations are shifting from traditional IT centric data architectures (where data is governed and managed centrally —to→ more federated models organized according to bussiness needs.

- This federated models are called Data mesh

- is a decentralized data architecture that organizes data by specific bussiness domains (e.g sales, HR.)

- data mesh architecture primarily supports organizing data into domains and enabling data consumers to filter and find content by domain

- It also enables federated governance, which means that some governance currently controlled at the tenant level can be delegated to domain-level control. enabling customised rules and restrictions for different domains.

- Domain is way of logically together all the data in an organization that is relevant to particular area or field.

- When a workspace is associated with the domain, all the items in the workspace are also associated with the domain, and they recieve a domain attribute as part of thier metadata.

Note Domain assignment doesnot affect item visibility or accessibility for tenant users.

Item discovery, visibility, and access depend on such things as workplace role and item permissions, but not domain assignment

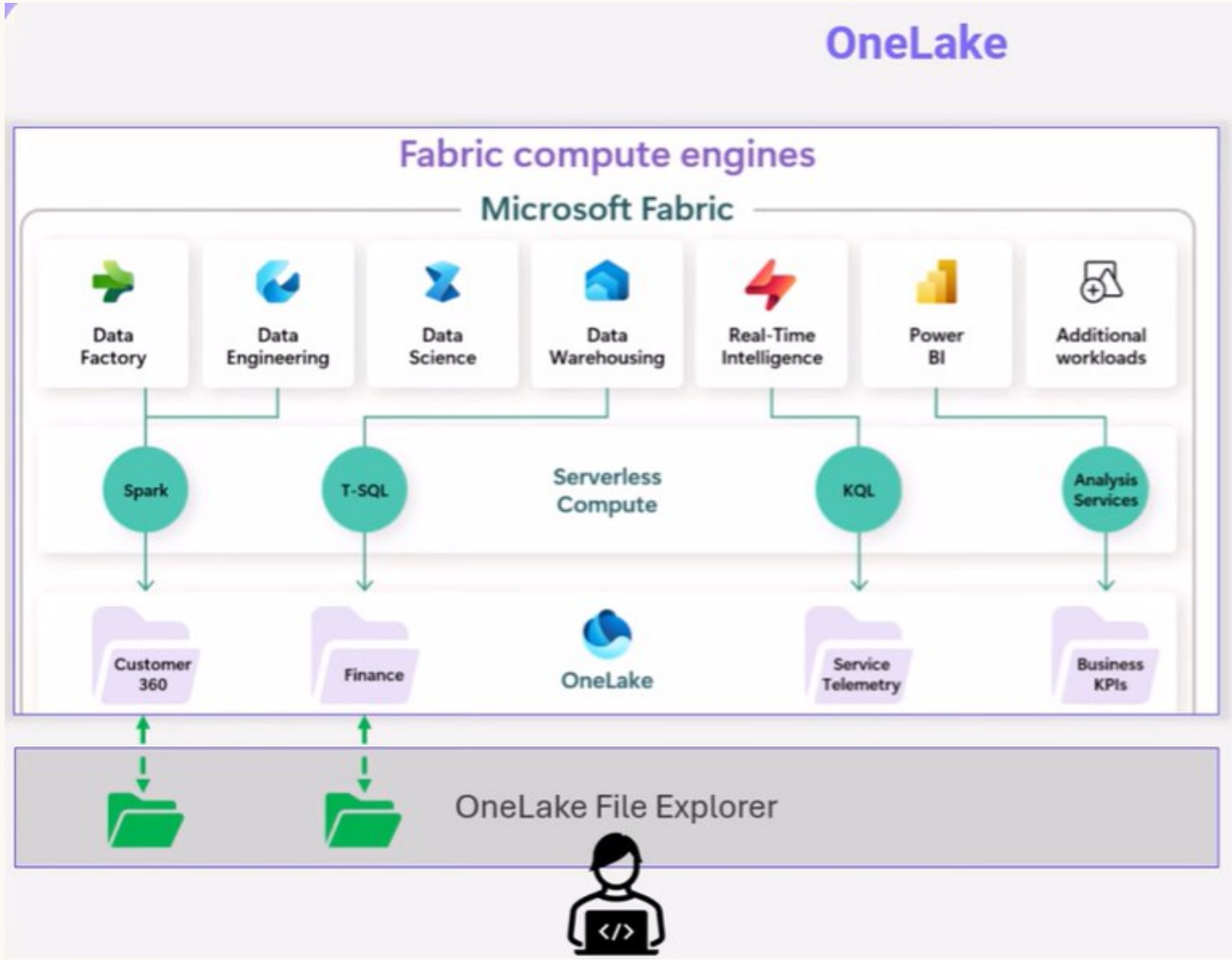
So, what can we currently do with Domains (by Role)?

Task	Fabric Admin	Domain Admin	Domain Contributor (Workspace Admin)
Create & delete domains	✓		
Appoint Domain Admins	✓		
View and edit all domains	✓		
Appoint Domain Contributors	✓	✓	
View and edit their own domains	✓	✓	
Override delegated tenant settings*	✓	✓	
Assign workspaces to domains (bulk, in Domain Settings)	✓	✓	
Assign workspaces to domains (singular, in Workspace Settings)	✓	✓	✓

* The following tenant-level settings can be delegated to the Domain-level:
- Domain-level default sensitivity label
- Certification Settings

Default domain

- is a domain that has been defined as the default domain for specified users and/or security group
- The system scans the organization's workspace, when it finds a workspace admin
  - (if) workspace is assigned to some domain → nothing happens
  - (else) it's assigned to default domain



The OneLake File Explorer makes **data in OneLake** accessible on your local Machine, much like OneDrive does for documents.

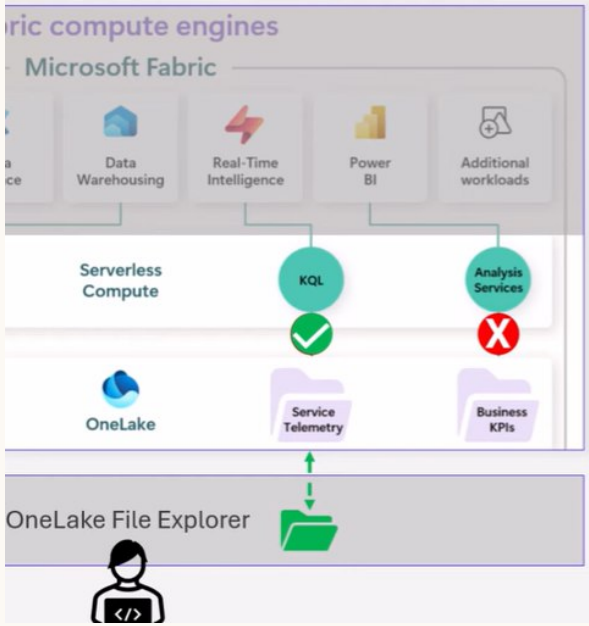
It provides a handy way for business users to get files into a Lakehouse, for example.

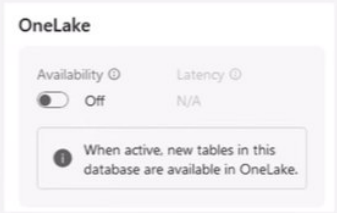
By default, Power BI semantic models and KQL databases do Not get written to OneLake (and ∴ Not written to OneLake file explorer.

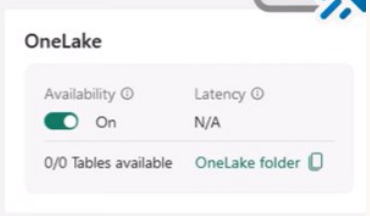
Note: there is a tenant-level setting to allow users to use OneLake File Explorer

but it can be changed

Enabling OneLake integration for KQL Database



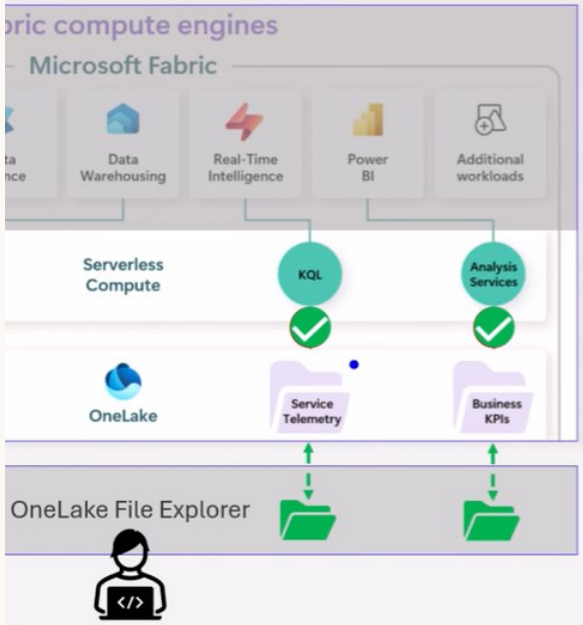




Step 1:
Within the UI of your KQL Database, turn on this 'Availability' button.

Note:
When you turn on OneLake integration in a KQL Database, it only works for *new* tables. It doesn't backfill existing data.

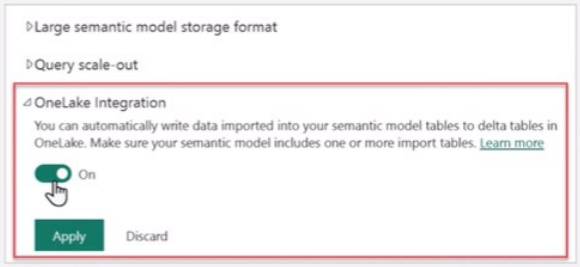
Enabling OneLake integration for Power BI Semantic Models



Note: only applies to Import mode semantic models/ tables

Step 1: Enable this functionality in the Tenant-level Admin Settings

Step 2: within the individual semantic model, enable OneLake Integration



Final note: when the delta tables for Semantic Models, and KQL Databases are written to OneLake like this, it opens up their use in other experiences.

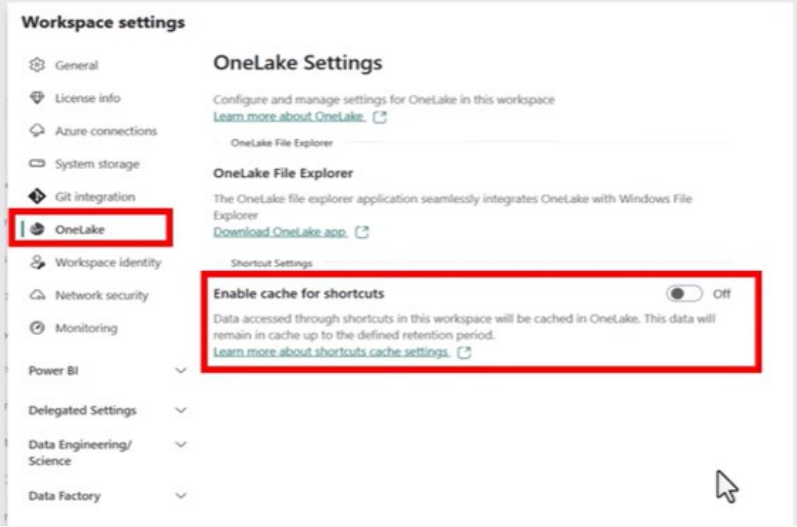
→ maybe not any more

Shortcuts & Shortcut Caching

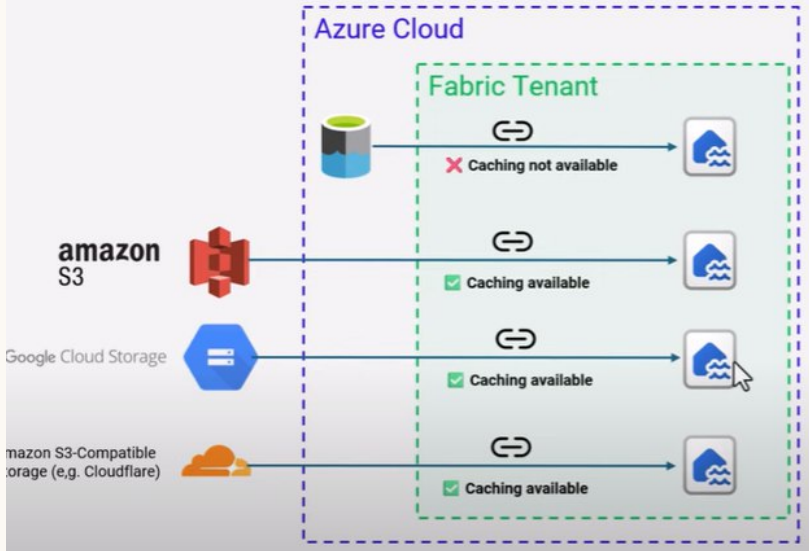
Shortcuts allow you to access external data in other cloud platforms from within Fabric, without the need for ETL.

Shortcut caching can be used to reduce egress costs associated with this cross-cloud data access.

Enable Shortcut Caching for data accessed through this Workspace:



Shortcuts & Shortcut Caching

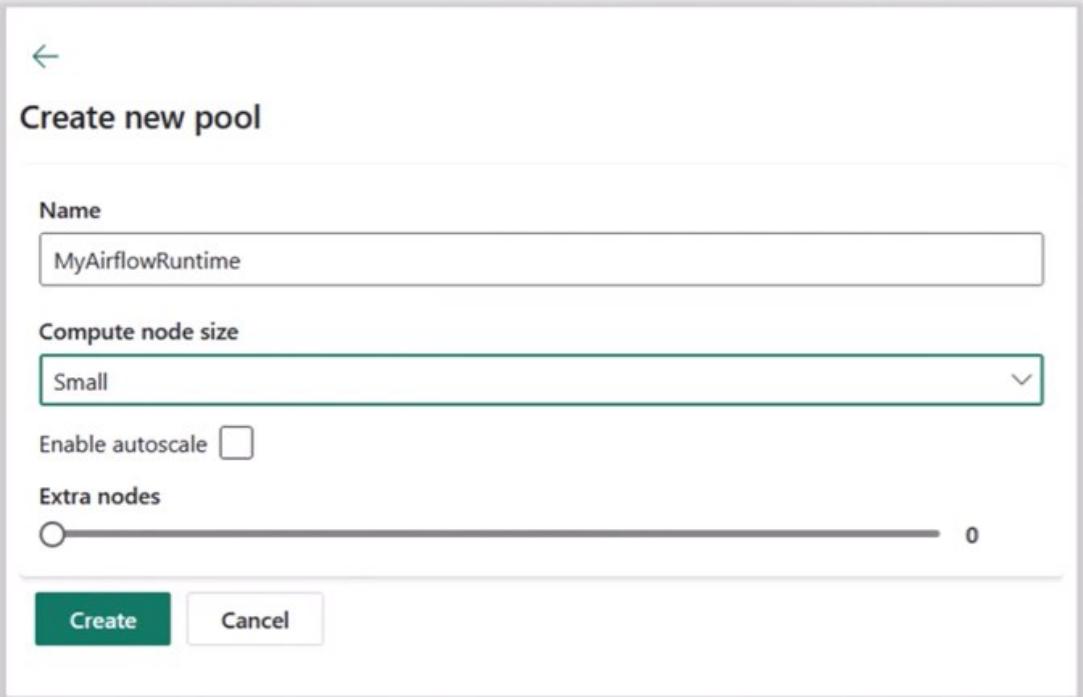


Key caching rules to be aware of:

24 hour rule: If a file hasn't been accessed for more than 24 hours, it's purged from the cache

1GB rule: Individual files greater than 1 GB in size aren't cached

Apache Airflow Jobs (aka Data Workflows)



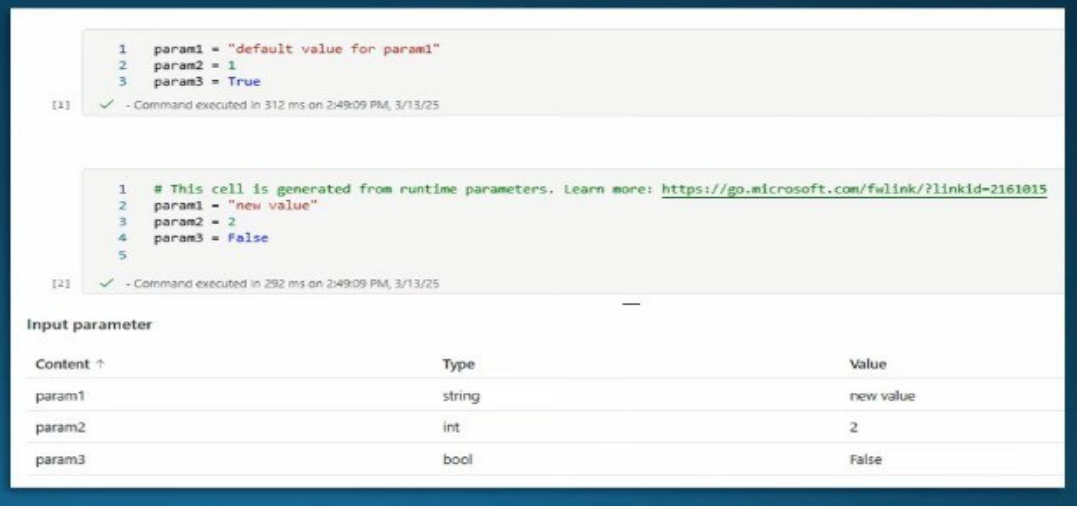
The Apache Airflow job (previously known as the Data Workflow) is a (preview) Fabric item that allows you to configure and runtime Apache Airflow DAGs (directed acyclic graphs).

These are orchestration jobs. They normally require a server (or hosted platform) to run a DAG, but Fabric provides a mechanism to run your DAG using a Pool.

Notebooks

Parameters

- One cell can be toggled as a parameter cell
  - Define default values as variables
- Different types of parameters are supported
  - String, Integer, Float and Boolean
- Execution engine adds a new cell beneath the parameter cell with input parameters to overwrite the default values



NotebookUtils (MsSparkUtils)

- Notebookutils
  - An integrated package in Microsoft Fabric Notebooks designed to streamline common tasks such as file system operations, environment variable access, notebook chaining, and secret management
- File System utilities (notebookutils.fs)
  - Interact with file systems like ADLS and OneLake
  - Create directories, move and delete files etc.
- Notebook utilities (notebookutils.notebook)
  - Run notebooks, set exit values, create notebooks etc.
- Credential utilities (notebookutils.credentials)
  - Get access tokens and manage secrets in an Azure Key Vault
- Other utilities
  - Lakehouse, Session and Runtime

```
notebookutils.fs.mkdirs("Files/new_folder")
notebookutils.fs.ls("Files/new_folder")
notebookutils.fs.rm('Files/new_folder', True)
```

```
notebookutils.notebook.run("nb_example_1", 90)
notebookutils.notebook.exit("value string")
```

```
notebookutils.credentials.getSecret('https://<name>.vault.azure.net/',\
'secret name')
```

```
notebookutils.lakehouse.list()
notebookutils.runtime.context
notebookutils.session.stop()
```

utility for dealing with lakehouse

utilities for getting some information and managing the current session

- Cell magic commands are on cell level and changes what the entire cell does

- Change the cell default language to anothe

line commands

- do operations on a single line of a cell

- There cannot be anything else in the cell than those commands.

```
// This cell is using Spark Scala
var scala_var = "Scala Variable"
```

```
%%sparkr
# This cell is using Spark R
r_var <- "R Variable"
```

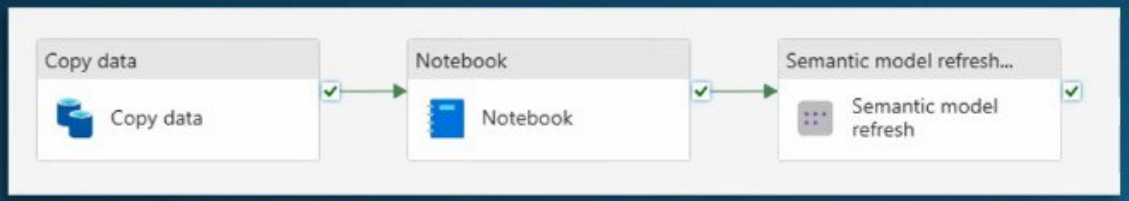
```
%%bash
# This cell is using Bash
bash_var="Bash Variable"
```

```
Note: you may need to restart the kernel to use updated packages.
Warning: PySpark kernel has been restarted to use updated packages.
```

```
%run nb_example_1
```

Running Notebooks

- Run manually
  - Run cell by cell or run the whole notebook
- Run notebooks from notebook
  - Use %run magic command
  - NotebookUtils library
    - notebookutils.notebook.run
    - notebookutils.notebook.runMultiple
- Run in a data pipeline




```
%run nb_example_1
```

```
notebookutils.notebook.run("nb_example_1", 90)
```

```
notebookutils.notebook.runMultiple(["nb_example_1", "nb_example_2"])
```

```
DAG = {
    "activities": [
        {
            "name": "nb_example_1",
            "path": "nb_example_1",
            "timeoutPerCellInSeconds": 90,
            "args": {"param_1": "some value"},
            "retry": 1,
            "retryIntervalInSeconds": 10
        },
        {
            "name": "nb_example_2",
            "path": "nb_example_2",
            "args": {"param_1": "some value"}
        },
        {
            "name": "nb_example_3",
            "path": "nb_example_3",
            "timeoutPerCellInSeconds": 90,
            "args": {"param_1": "some value"},
            "retry": 1,
            "retryIntervalInSeconds": 10,
            "dependencies": ["nb_example_1","nb_example_2"]
        }
    ],
    "timeoutInSeconds": 300,
    "concurrency": 3
}
exit_values_dag = notebookutils.notebook.runMultiple(DAG, {"displayDAGViaGraphviz": True})
```

run → run 1 notebook

runMultiple → run multiple notebooks in any order.

→ If we use dag with it, we can notebooks in specific order we need