**USE APACHE SPARK IN MS FABRIC**

**Spark pools**

Consists of compute nodes that distribute data processing tasks.

1. A ***head*** node in a Spark pool coordinates distributed processes through a *driver* program.
2. The pool includes multiple ***worker*** nodes on which *executor* processes perform the actual data processing tasks.

A diagram of a server

AI-generated content may be incorrect.

You can configure the starter pool to optimize the nodes it contains in accordance with your specific workload needs or cost constraints.

Additionally, you can create custom Spark pools with specific node configurations that support your particular data processing needs.

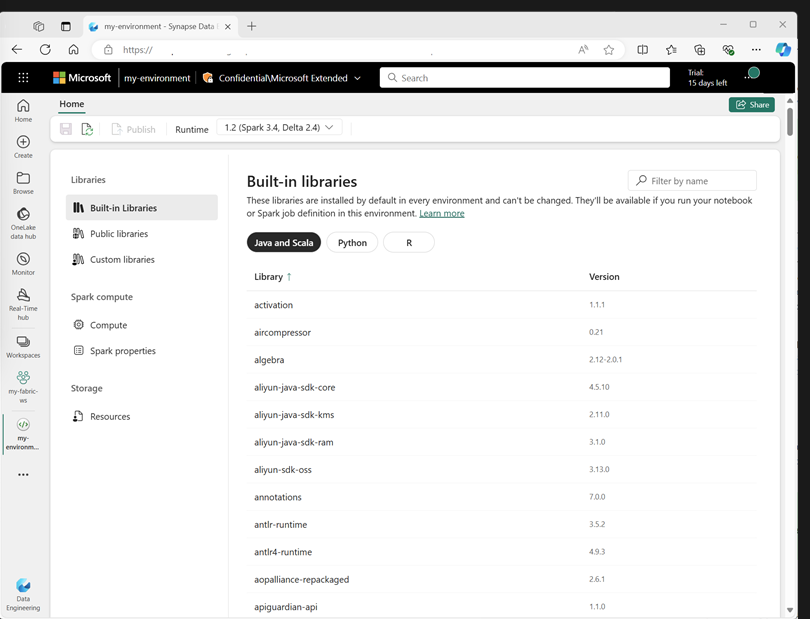
**Specific config settings for spark pools:**

|  |  |
| --- | --- |
| Node family | The type of virtual machines used for the Spark cluster nodes. In most cases, memory optimized nodes provide optimal performance. |
| Autoscale | Whether or not to automatically provision nodes as needed, and if so, the initial and maximum number of nodes to be allocated to the pool. |
| Dynamic allocation | Whether or not to dynamically allocate executor processes on the worker nodes based on data volumes. |

If you create one or more custom Spark pools in a workspace, you can set one of them (or the starter pool) as the default pool to be used if a specific pool is not specified for a given Spark job.

**Runtimes and environments**

* Version determination
* Install libs
* Each env -> runtime version; libraries installed



When creating an environment, you can:

* Specify the Spark runtime it should use.
* View the built-in libraries that are installed in every environment.
* Install specific public libraries from the Python Package Index (PyPI).
* Install custom libraries by uploading a package file.
* Specify the Spark pool that the environment should use.
* Specify Spark configuration properties to override default behavior.
* Upload resource files that need to be available in the environment.

**Native Execution engine:**

* Vectorized processing engine 🡪 runs spark operations directly on LK infrastructure. Improve query performance on large data sets in Parquet/ delta file formats.
* Enable in 🡪 environment level or notebook level
* **spark.native.enabled**: true
* **spark.shuffle.manager**: org.apache.spark.shuffle.sort.ColumnarShuffleManager

In beginning of your code:

**%%configure**

**{**

**"conf": {**

**"spark.native.enabled": "true",**

**"spark.shuffle.manager": "org.apache.spark.shuffle.sort.ColumnarShuffleManager"**

**}**

**}**

**High concurrency mode**

**Automatic MLFlow logging**

**Spark administration for a Fabric capacity**

**Spark job definition**

If you want to use Spark to ingest and transform data as part of an automated process, you can define a Spark job to run a script on-demand or based on a schedule.

A screenshot of a computer

AI-generated content may be incorrect.

To configure a Spark job, create a Spark Job Definition in your workspace and specify the script it should run. You can also specify a reference file (for example, a Python code file containing definitions of functions that are used in your script) and a reference to a specific lakehouse containing data that the script processes.

**Parquet file:**

The Parquet format is typically preferred for data files that you will use for further analysis or ingestion into an analytical store. Parquet is a very efficient format that is supported by most large scale data analytics systems. In fact, sometimes your data transformation requirement may simply be to convert data from another format (such as CSV) to Parquet!

**Partitioning the output file**

Partitioning is an optimization technique that enables Spark to maximize performance across the worker nodes. More performance gains can be achieved when filtering data in queries by eliminating unnecessary disk IO.

**Load partitioned data**

When reading partitioned data into a dataframe, you can load data from any folder within the hierarchy by specifying explicit values or wildcards for the partitioned fields.

**Work with data using Spark SQL**

df.createOrReplaceTempView("products\_view")

df.write.format("delta").saveAsTable("products")

* A ***view*** is **temporary**, meaning that it's automatically deleted at the end of the current session. You can also create *tables* that are persisted in the catalog to define a database that can be queried using Spark SQL.
* **Tables** are **metadata** **structures** that store their underlying data in the storage location associated with the catalog. You can create an empty table by using the **spark.catalog.createTable** method, or you can save a **dataframe** as a **table** by using its **saveAsTable** method. Deleting a managed table also deletes its underlying data.
* Additionally, you can create ***external*** tables by using the **spark.catalog.createExternalTable**method. External tables define **metadata** in the catalog but get their underlying data from an external storage location; typically a folder in the **Files** storage area of a **lakehouse**. Deleting an external table doesn't delete the underlying data.

**Using the Spark SQL API to query data**

bikes\_df = spark.sql("SELECT ProductID, ProductName, ListPrice \

FROM products \

WHERE Category IN ('Mountain Bikes', 'Road Bikes')")

display(bikes\_df)

**Visualize data in a Spark notebook**

**Using graphics packages in code**

from matplotlib import pyplot as plt

# Get the data as a Pandas dataframe

data = spark.sql("SELECT Category, COUNT(ProductID) AS ProductCount \

FROM products \

GROUP BY Category \

ORDER BY Category").toPandas()

# Clear the plot area

plt.clf()

# Create a Figure

fig = plt.figure(figsize=(12,8))

# Create a bar plot of product counts by category

plt.bar(x=data['Category'], height=data['ProductCount'], color='orange')

# Customize the chart

plt.title('Product Counts by Category')

plt.xlabel('Category')

plt.ylabel('Products')

plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.7)

plt.xticks(rotation=70)

# Show the plot area

plt.show()

A screenshot of a computer

AI-generated content may be incorrect.