Data Engineering documentation in Microsoft Fabric

Data engineering in Microsoft Fabric enables users to design, build, and maintain infrastructures and systems that enable their organizations to collect, store, process, and analyze large volumes of data.

About Data Engineering

OVERVIEW

What is Data engineering?

What is a lakehouse?

Workspace roles and permissions

Get started with lakehouse

GET STARTED

Create a lakehouse

Navigate the lakehouse explorer

HOW-TO GUIDE

Get data into lakehouse using notebook

Get started with notebooks

GET STARTED

Microsoft Fabric notebooks

HOW-TO GUIDE

Build and manage notebooks

Explore lakehouse data with a notebook

Visualize notebooks

Delta Lake

CONCEPT

Lakehouse and Delta tables

HOW-TO GUIDE

Delta optimization and V-Order

Load to tables

Apache Spark

CONCEPT

Apache Spark job definition

Spark monitoring overview

Apache Spark advisor

GET STARTED

Create a Spark job definition

HOW-TO GUIDE

Schedule and run a Spark job definition

Monitor a Spark application

Manage Apache Spark libraries

Spark utilities for file management tasks

Configure capacity settings

Lakehouse end-to-end tutorial

CONCEPT

Lakehouse scenario overview

TUTORIAL

Step 1 - Create a workspace

Step 2 - Create a lakehouse

Step 3 - Ingest data into lakehouse

Step 4 - Transform data into delta tables

Step 5 - Build a report

Step 6 - Clean up resources

VS Code integration

HOW-TO GUIDE

Use notebooks from VS Code

Use Spark job definitions from VS Code

Explore lakehouse from VS Code

What is Data engineering in Microsoft Fabric?

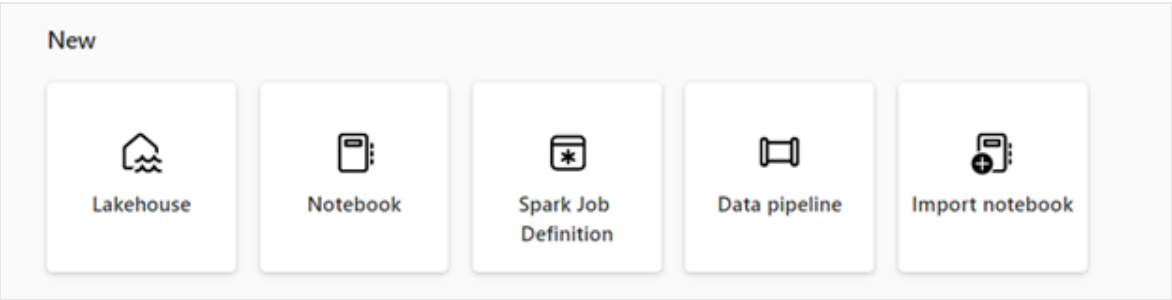
Article • 05/23/2023

Data engineering in Microsoft Fabric enables users to design, build, and maintain infrastructures and systems that enable their organizations to collect, store, process, and analyze large volumes of data.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Microsoft Fabric provides various data engineering capabilities to ensure that your data is easily accessible, well-organized, and of high-quality. From the data engineering homepage, you can:

- Create and manage your data using a lakehouse
- Design pipelines to copy data into your lakehouse
- Use Spark Job definitions to submit batch/streaming job to Spark cluster
- Use notebooks to write code for data ingestion, preparation, and transformation



Lakehouse

Lakehouses are data architectures that allow organizations to store and manage structured and unstructured data in a single location, using various tools and frameworks to process and analyze that data. This can include SQL-based queries and analytics, as well as machine learning and other advanced analytics techniques.

Apache Spark job definition

Spark job definitions are set of instructions that define how to execute a job on a Spark cluster. It includes information such as the input and output data sources, the transformations, and the configuration settings for the Spark application. Spark job Definition allows you to submit batch/streaming job to Spark cluster, apply different transformation logic to the data hosted on your Lakehouse along with many other things.

Notebook

Notebooks are an interactive computing environment that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. They allow users to write and execute code in various programming languages, including Python, R, and Scala and are used for data ingestion, preparation, analysis, and other data-related tasks.

Data pipeline

Data pipelines are a series of steps that are used to collect, process, and transform data from its raw form to a format that can be used for analysis and decision-making. They're a critical component of data engineering, as they provide a way to move data from its source to its destination in a reliable, scalable, and efficient way.

Next steps

Get started with the Data Engineering experience:

- Learn more about Lakehouse, see What is a Lakehouse?.
- To get started with a Lakehouse, see Creating a Lakehouse.
- Learn more about Apache Spark job definitions, see What is an Apache Spark job definition?.
- To get started with an Apache Spark job definition, see Creating a Apache Spark job definition.
- Learn more about Notebooks, see Author and execute the notebook.
- To get started with Pipelines copy activity, see How to copy data using copy activity.

Microsoft Fabric decision guide: copy activity, dataflow, or Spark

Article • 05/23/2023

Use this reference guide and the example scenarios to help you in deciding whether you need a copy activity, a dataflow, or Spark for your workloads using Microsoft Fabric.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Copy activity, dataflow, and Spark properties

	Pipeline copy activity	Dataflow Gen 2	Spark
Use case	Data lake and data warehouse migration, data ingestion, lightweight transformation	Data ingestion, data transformation, data wrangling, data profiling	Data ingestion, data transformation, data processing, data profiling
Primary developer persona	Data engineer, data integrator	Data engineer, data integrator, business analyst	Data engineer, data scientist, data developer
Primary developer skill set	ETL, SQL, JSON	ETL, M, SQL	Spark (Scala, Python, Spark SQL, R)
Code written	No code, low code	No code, low code	Code
Data volume	Low to high	Low to high	Low to high
Development interface	Wizard, canvas	Power query	Notebook, Spark job definition

	Pipeline copy activity	Dataflow Gen 2	Spark
Sources	30+ connectors	150+ connectors	Hundreds of Spark libraries
Destinations	18+ connectors	Lakehouse, Azure SQL database, Azure Data explorer, Azure Synapse analytics	Hundreds of Spark libraries
Transformation complexity	Low: lightweight - type conversion, column mapping, merge/split files, flatten hierarchy	Low to high: 300+ transformation functions	Low to high: support for native Spark and open-source libraries

Review the following three scenarios for help with choosing how to work with your data in Fabric.

Scenario1

Leo, a data engineer, needs to ingest a large volume of data from external systems, both on-premises and cloud. These external systems include databases, file systems, and APIs. Leo doesn't want to write and maintain code for each connector or data movement operation. He wants to follow the medallion layers best practices, with bronze, silver, and gold. Leo doesn't have any experience with Spark, so he prefers the drag and drop UI as much as possible, with minimal coding. And he also wants to process the data on a schedule.

The first step is to get the raw data into the bronze layer lakehouse from Azure data resources and various third party sources (like Snowflake Web, REST, AWS S3, GCS, etc.). He wants a consolidated lakehouse, so that all the data from various LOB, on-premises, and cloud sources reside in a single place. Leo reviews the options and selects **pipeline copy activity** as the appropriate choice for his raw binary copy. This pattern applies to both historical and incremental data refresh. With copy activity, Leo can load Gold data to a data warehouse with no code if the need arises and pipelines provide high scale data ingestion that can move petabyte-scale data. Copy activity is the best low-code and no-code choice to move petabytes of data to lakehouses and warehouses from varieties of sources, either ad-hoc or via a schedule.

Scenario2

Mary is a data engineer with a deep knowledge of the multiple LOB analytic reporting requirements. An upstream team has successfully implemented a solution to migrate multiple LOB's historical and incremental data into a common lakehouse. Mary has been tasked with cleaning the data, applying business logics, and loading it into multiple destinations (such as Azure SQL DB, ADX, and a lakehouse) in preparation for their respective reporting teams.

Mary is an experienced Power Query user, and the data volume is in the low to medium range to achieve desired performance. Dataflows provide no-code or low-code interfaces for ingesting data from hundreds of data sources. With dataflows, you can transform data using 300+ data transformation options, and write the results into multiple destinations with an easy to use, highly visual user interface. Mary reviews the options and decides that it makes sense to use **Dataflow Gen 2** as her preferred transformation option.

Scenario3

Adam is a data engineer working for a large retail company that uses a lakehouse to store and analyze its customer data. As part of his job, Adam is responsible for building and maintaining the data pipelines that extract, transform, and load data into the lakehouse. One of the company's business requirements is to perform customer review analytics to gain insights into their customers' experiences and improve their services.

Adam decides the best option is to use **Spark** to build the extract and transformation logic. Spark provides a distributed computing platform that can process large amounts of data in parallel. He writes a Spark application using Python or Scala, which reads structured, semi-structured, and unstructured data from OneLake for customer reviews and feedback. The application cleanses, transforms, and writes data to Delta tables in the lakehouse. The data is then ready to be used for downstream analytics.

Next steps

- How to copy data using copy activity
- Quickstart: Create your first dataflow to get and transform data
- How to create an Apache Spark job definition in Fabric

Microsoft Fabric decision guide: data warehouse or lakehouse

Article • 05/23/2023

Use this reference guide and the example scenarios to help you choose between the data warehouse or a lakehouse for your workloads using Microsoft Fabric.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Data warehouse and lakehouse properties

	Data warehouse	Lakehouse	Power BI Datamart
Data volume	Unlimited	Unlimited	Up to 100 GB
Type of data	Structured	Unstructured, semi-structured, structured	Structured
Primary developer persona	Data warehouse developer, SQL engineer	Data engineer, data scientist	Citizen developer
Primary developer skill set	SQL	Spark (Scala, PySpark, Spark SQL, R)	No code, SQL
Data organized by	Databases, schemas, and tables	Folders and files, databases and tables	Database, tables, queries
Read operations	Spark, T-SQL	Spark, T-SQL	Spark, T-SQL, Power BI
Write operations	T-SQL	Spark (Scala, PySpark, Spark SQL, R)	Dataflows, T-SQL

	Data warehouse	Lakehouse	Power BI Datamart
Multi-table transactions	Yes	No	No
Primary development interface	SQL scripts	Spark notebooks, Spark job definitions	Power BI
Security	Object level (table, view, function, stored procedure, etc.), column level, row level, DDL/DML	Row level, table level (when using T-SQL), none for Spark	Built-in RLS editor
Access data via shortcuts	Yes (indirectly through the lakehouse)	Yes	No
Can be a source for shortcuts	Yes (tables)	Yes (files and tables)	No
Query across items	Yes, query across lakehouse and warehouse tables	Yes, query across lakehouse and warehouse tables; query across lakehouses (including shortcuts using Spark)	No

Scenarios

Review these scenarios for help with choosing between using a lakehouse or a data warehouse in Fabric.

Scenario 1

Susan, a professional developer, is new to Microsoft Fabric. They are ready to get started cleaning, modeling, and analyzing data but need to decide to build a data warehouse or a lakehouse. After review of the details in the previous table, the primary decision points are the available skill set and the need for multi-table transactions.

Susan has spent many years building data warehouses on relational database engines, and is familiar with SQL syntax and functionality. Thinking about the larger team, the primary consumers of this data are also skilled with SQL and SQL analytical tools. Susan decides to use a **data warehouse**, which allows the team to interact primarily with T-SQL, while also allowing any Spark users in the organization to access the data.

Scenario 2

Rob, a data engineer, needs to store and model several terabytes of data in Fabric. The team has a mix of PySpark and T-SQL skills. Most of the team running T-SQL queries are consumers, and therefore don't need to write INSERT, UPDATE, or DELETE statements. The remaining developers are comfortable working in notebooks, and because the data is stored in Delta, they're able to interact with a similar SQL syntax.

Rob decides to use a **lakehouse**, which allows the data engineering team to use their diverse skills against the data, while allowing the team members who are highly skilled in T-SQL to consume the data.

Scenario 3

Ash, a citizen developer, is a Power BI developer. They're familiar with Excel, Power BI, and Office. They need to build a data product for a business unit. They know they don't quite have the skills to build a data warehouse or a lakehouse, and those seem like too much for their needs and data volumes. They review the details in the previous table and see that the primary decision points are their own skills and their need for a self service, no code capability, and data volume under 100 GB.

Ash works with business analysts familiar with Power BI and Microsoft Office, and knows that they already have a Premium capacity subscription. As they think about their larger team, they realize the primary consumers of this data may be analysts, familiar with no-code and SQL analytical tools. Ash decides to use a **Power BI datamart**, which allows the team to interact build the capability fast, using a no-code experience. Queries can be executed via Power BI and T-SQL, while also allowing any Spark users in the organization to access the data as well.

Next steps

- What is data warehousing in Microsoft Fabric?
- Create a warehouse in Microsoft Fabric
- Create a lakehouse in Microsoft Fabric
- Introduction to Power BI datamarts

CSV file upload to Delta for Power BI reporting

Article • 05/23/2023

Microsoft Fabric Lakehouse is a data architecture platform for storing, managing, and analyzing structured and unstructured data in a single location.

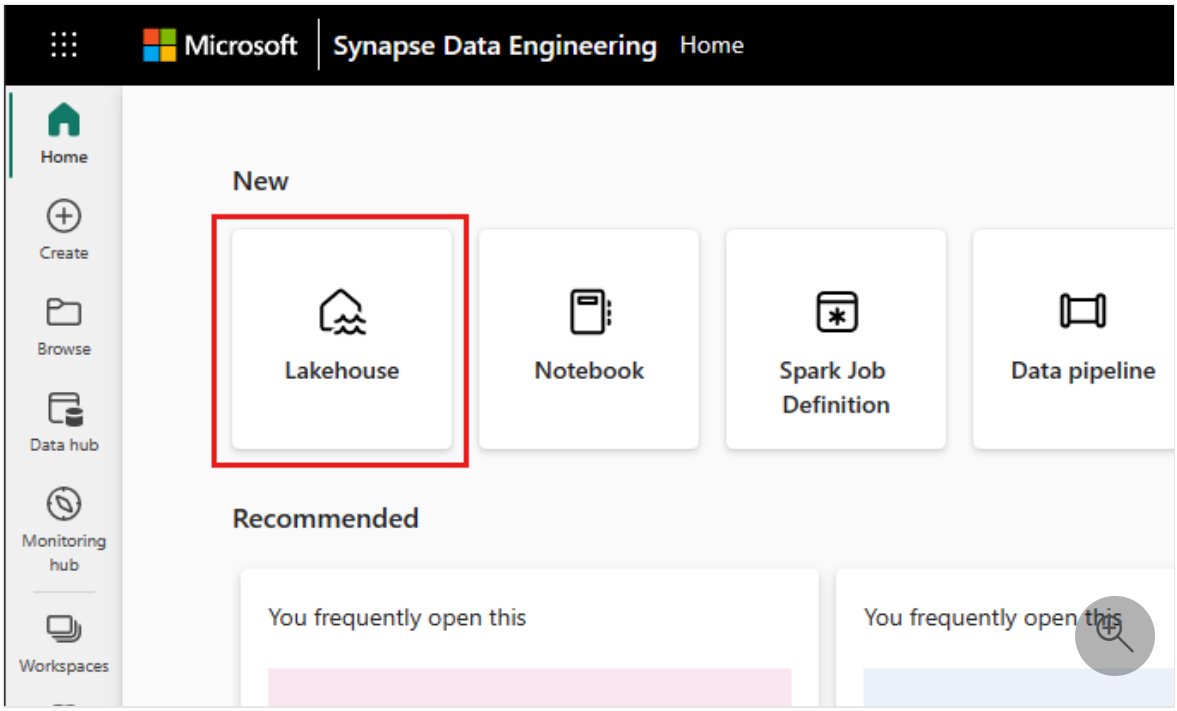
> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this tutorial you learn to:

- Upload a CSV file to a Lakehouse
- Convert the file to a Delta table
- Generate a Dataset and create a Power BI report

Create a Lakehouse and get a CSV file ready

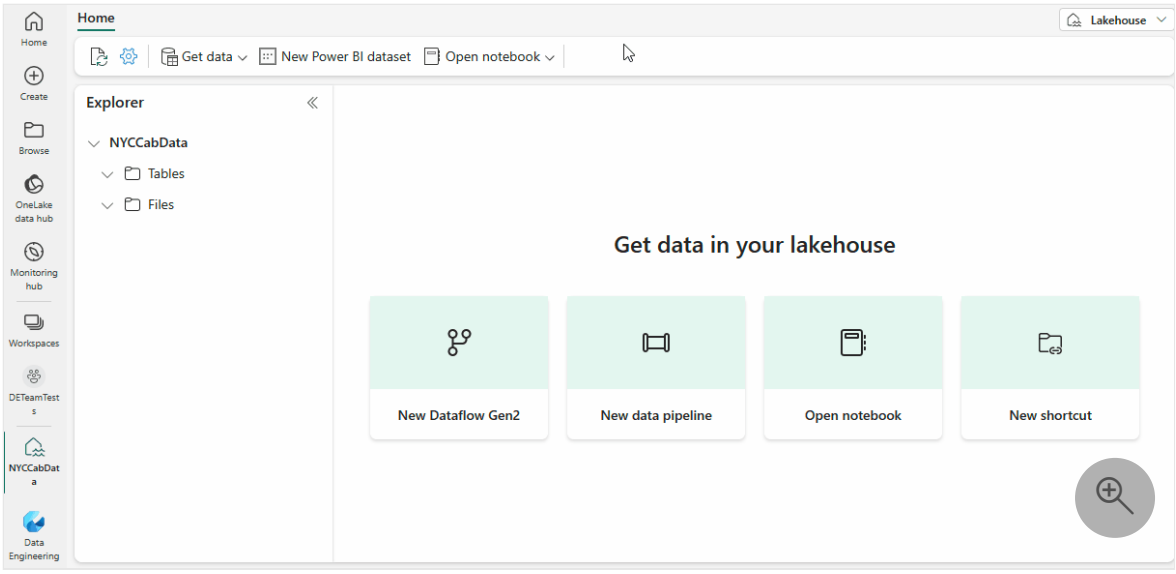
1. In Microsoft Fabric, select **Synapse Data Engineering** experience
2. Make sure you are in desired workspace or select/create one
3. Select **Lakehouse** icon under New section in the main mage



4. Enter name of your Lakehouse
5. Select **Create**
6. Download the "Taxi Zone Lookup Table" CSV file from the TLC Trip Record Data website, and save to a location in your computer.

Upload a CSV file to the Lakehouse

1. Create the `TaxiData` folder under the `Files` section of your Lakehouse.
2. Upload the file to the folder, by using the **Upload file** item in the folder contextual menu.
3. Once uploaded, select the folder to see its content.
4. Rename the file to remove special characters, in this example, remove the '+' character. To see the full list of special characters, read the Load to Delta Lake tables article.



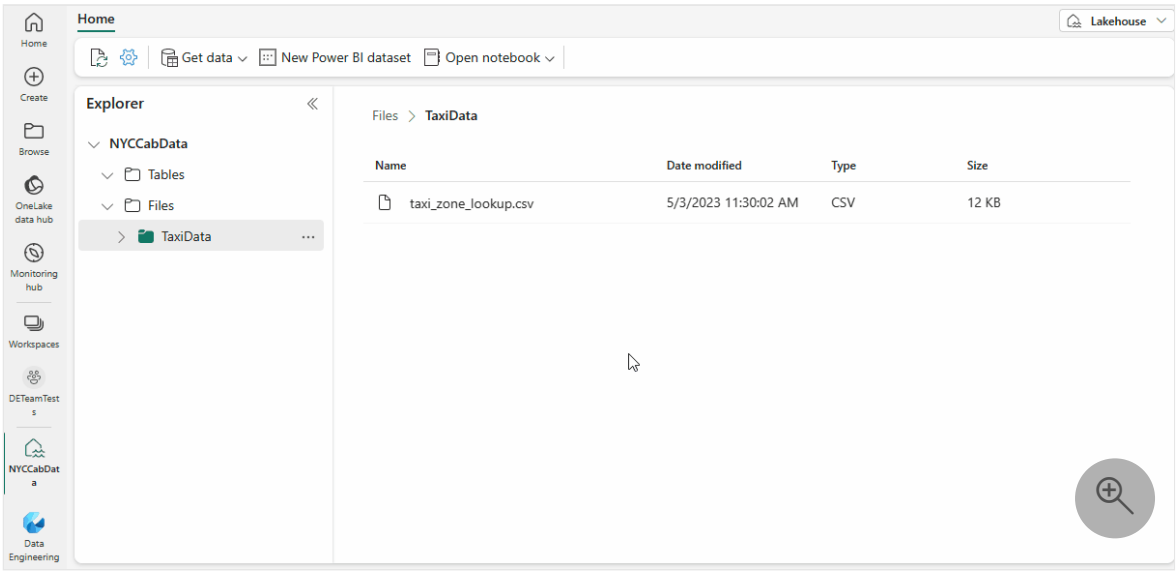
Load the file to a Delta table

1. Right-click or use the ellipsis on the CSV file to access the contextual menu, and select **Load to Delta**.

2. The load to tables user interface shows up with the suggested table name. Real time validations on special characters apply during typing.

3. Select **Confirm** to execute the load.

4. The table now shows up in the lakehouse explorer, expand the table to see the columns and its types. Select the table to see a preview.

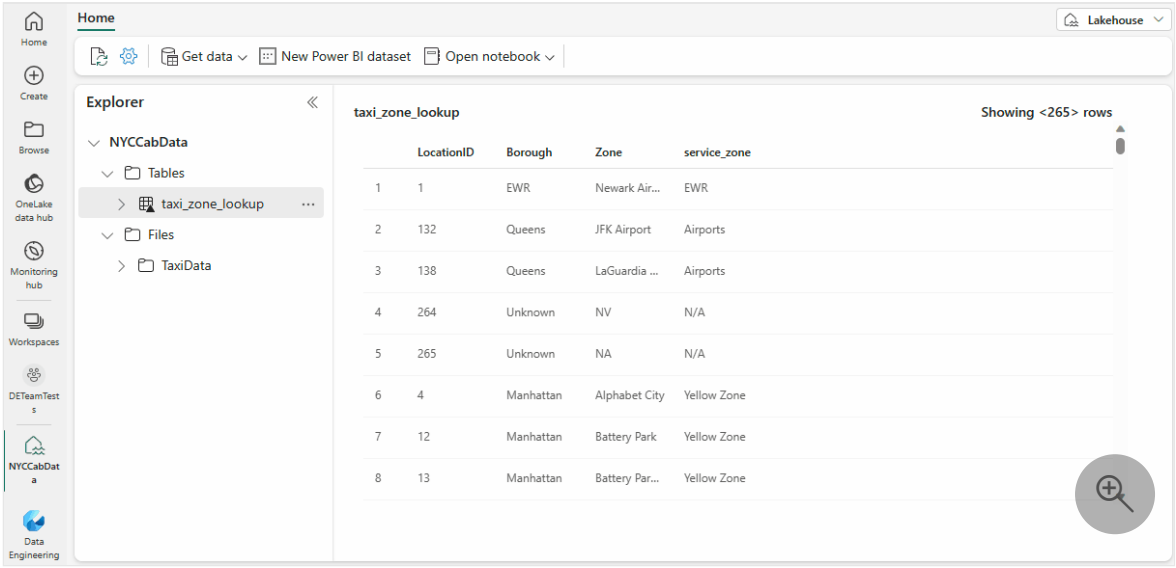


> **Note**
>
> If the table already exists, different **load mode** options are show. **Overwrite** will drop and recreate the table. **Append** will insert all CSV content as new data. For an

in-depth guide on the **Load to Tables** feature, read the Load to Tables article.

Generate a dataset and create a Power BI report

1. Select **New Power BI dataset** on the Lakehouse ribbon.
2. Select the table to be added to the dataset model, select the **Confirm** button.
3. On the dataset editing experience, you are able to define relationships between multiple tables, and also apply data types normalization and DAX transformations to the data if desired.
4. Select **New report** on the ribbon.
5. Use the report builder experience to design a Power BI report.



Next steps

- Load to Delta Lake tables
- What is Delta Lake?

Get streaming data into lakehouse and access with SQL endpoint

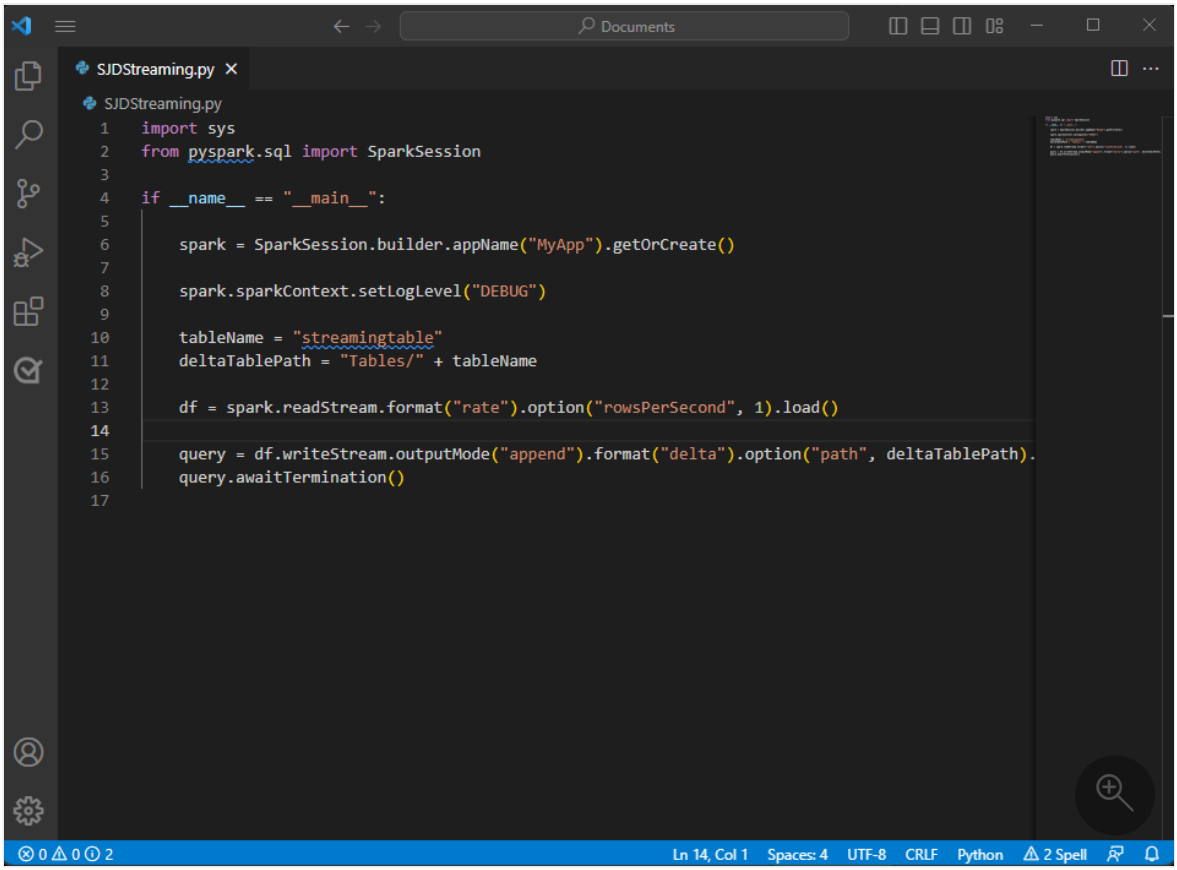
Article • 05/23/2023

This quickstart explains how to create a Spark Job Definition that contains Python code with Spark Structured Streaming to land data in a lakehouse and then serve it through a SQL endpoint. After completing this quickstart, you'll have a Spark Job Definition that runs continuously and the SQL endpoint can view the incoming data.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Create a Python script

1. Use the following Python code that uses Spark structured streaming to get data in a lakehouse table.

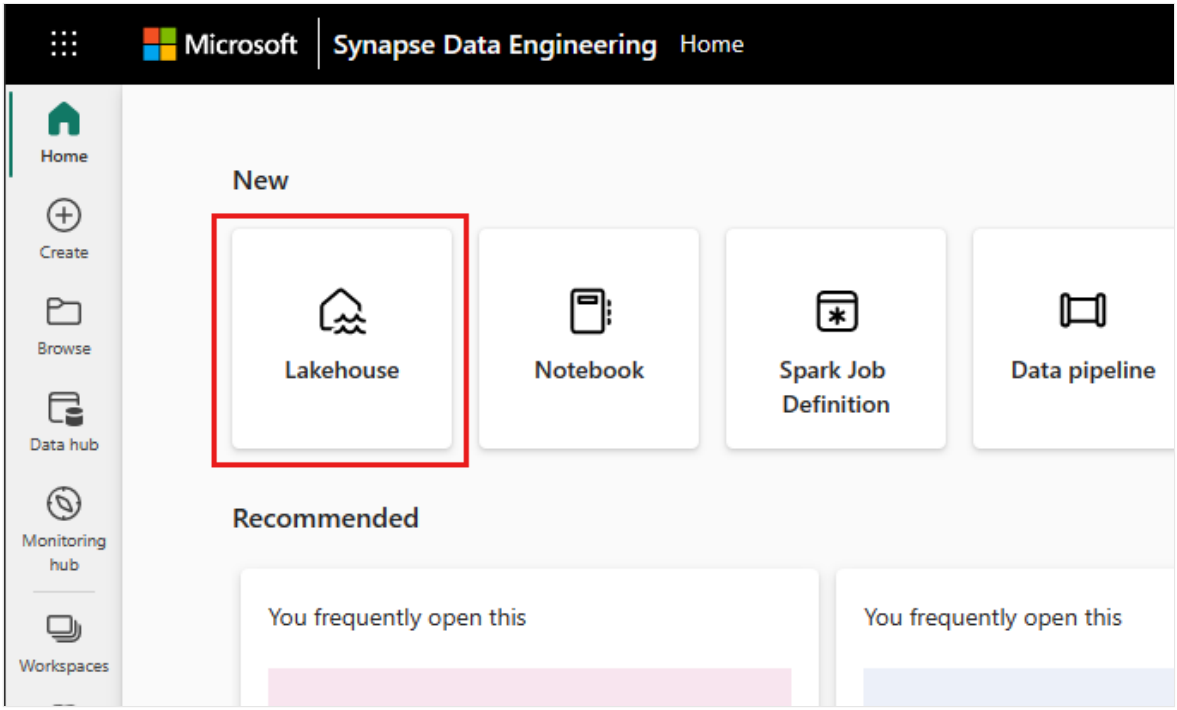


2. Save your script as Python file (.py) in your local computer.

Create a lakehouse

Use the following steps to create a lakehouse:

1. In Microsoft Fabric, select the **Synapse Data Engineering** experience.
2. Navigate to your desired workspace or create a new one if needed.
3. To create a lakehouse, select the **Lakehouse** icon under the **New** section in the main pane.

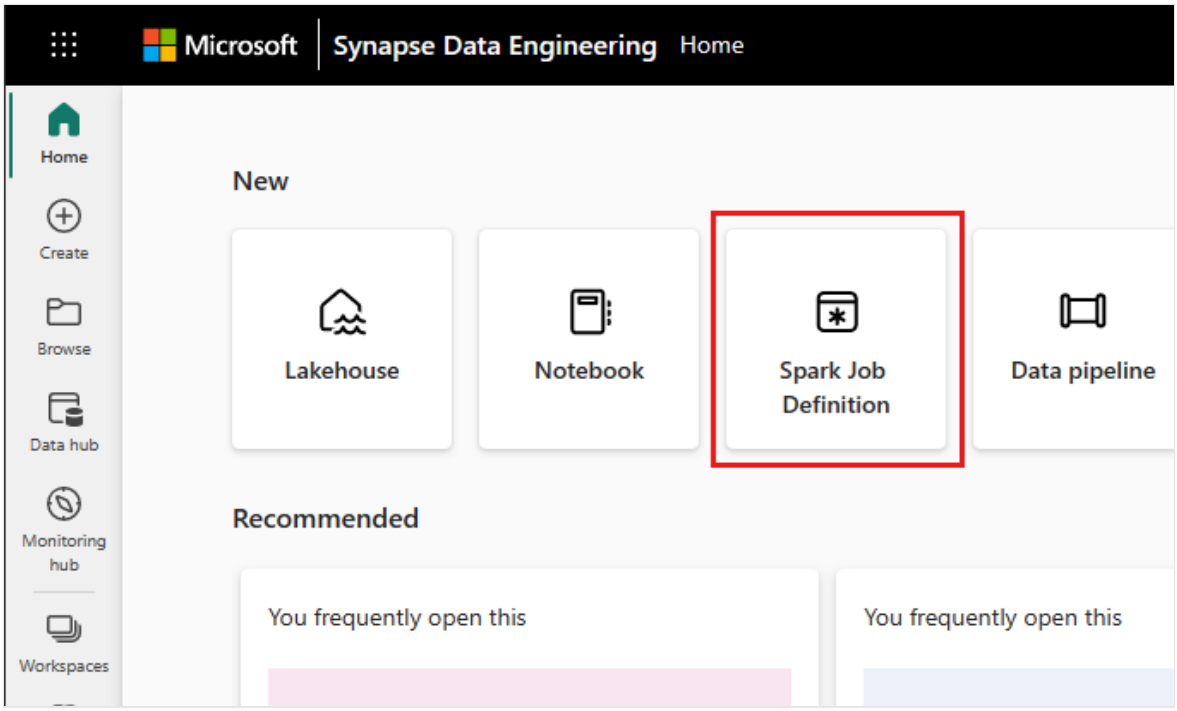


4. Enter name of your lakehouse and select **Create**.

Create a Spark Job Definition

Use the following steps to create a Spark Job Definition:

1. From the same workspace where you created a lakehouse, select the **Create** icon from the left menu.
2. Under "Data Engineering", select **Spark Job Definition**.



3. Enter name of your Spark Job Definition and select **Create**.

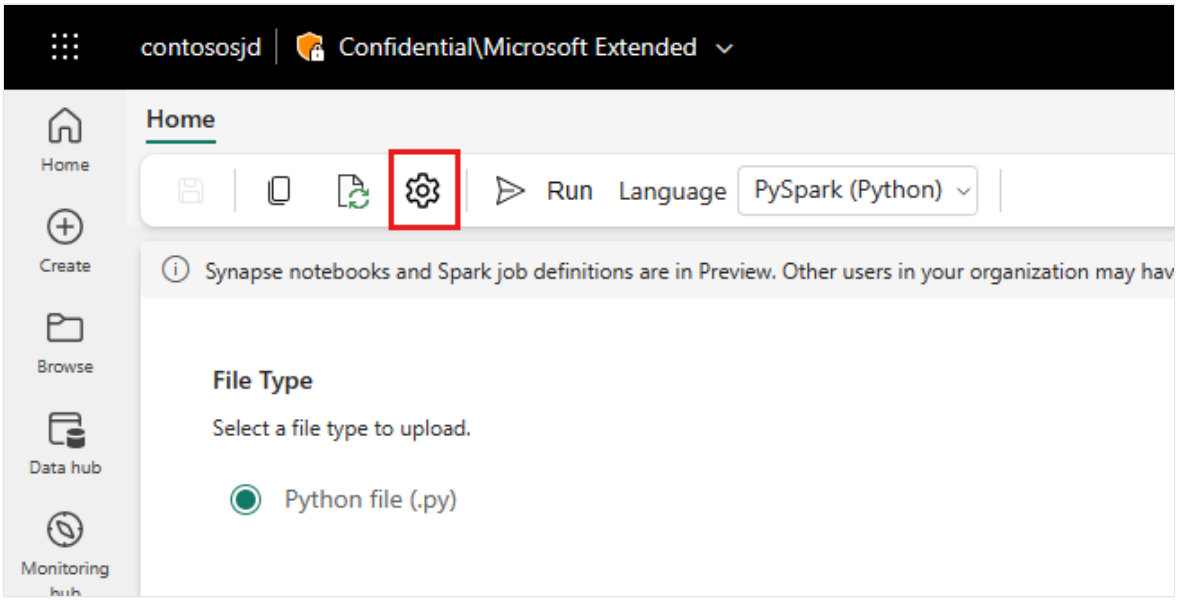
4. Select **Upload** and select the Python file you created in the previous step.

5. Under **Lakehouse Reference** choose the lakehouse you created.

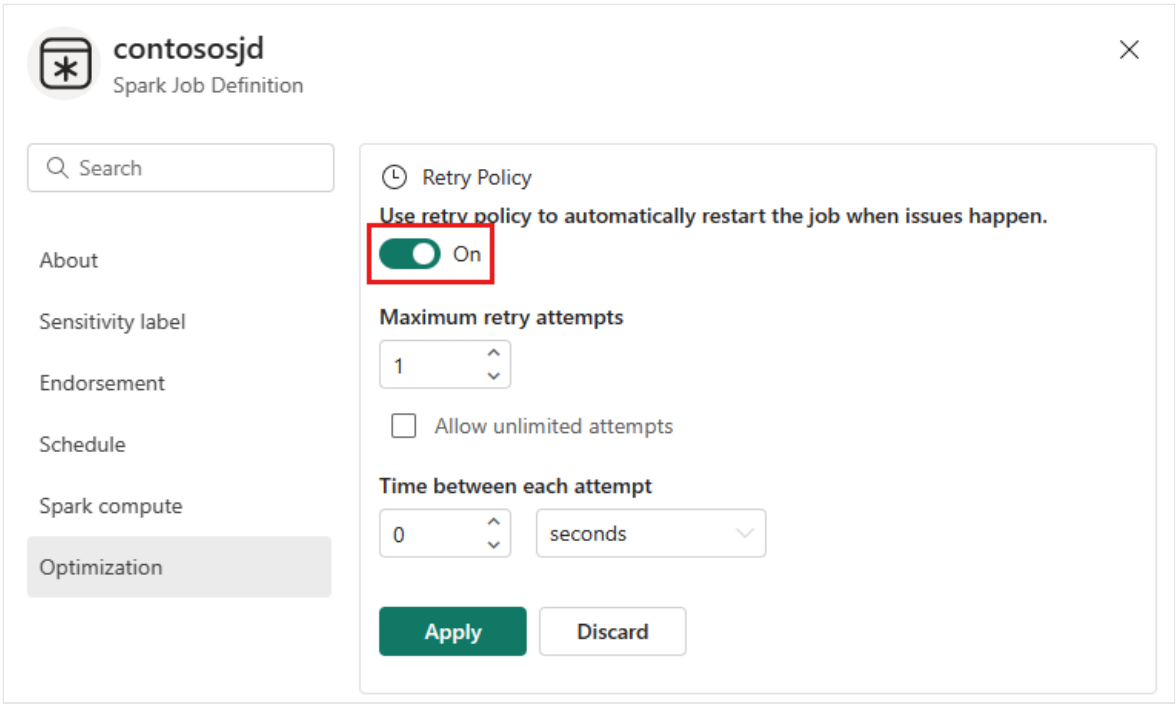
Set Retry policy for Spark Job Definition

Use the following steps to set the retry policy for your Spark job definition:

1. From the top menu, select the **Setting** icon.



2. Open the **Optimization** tab and set **Retry Policy** trigger **On**.

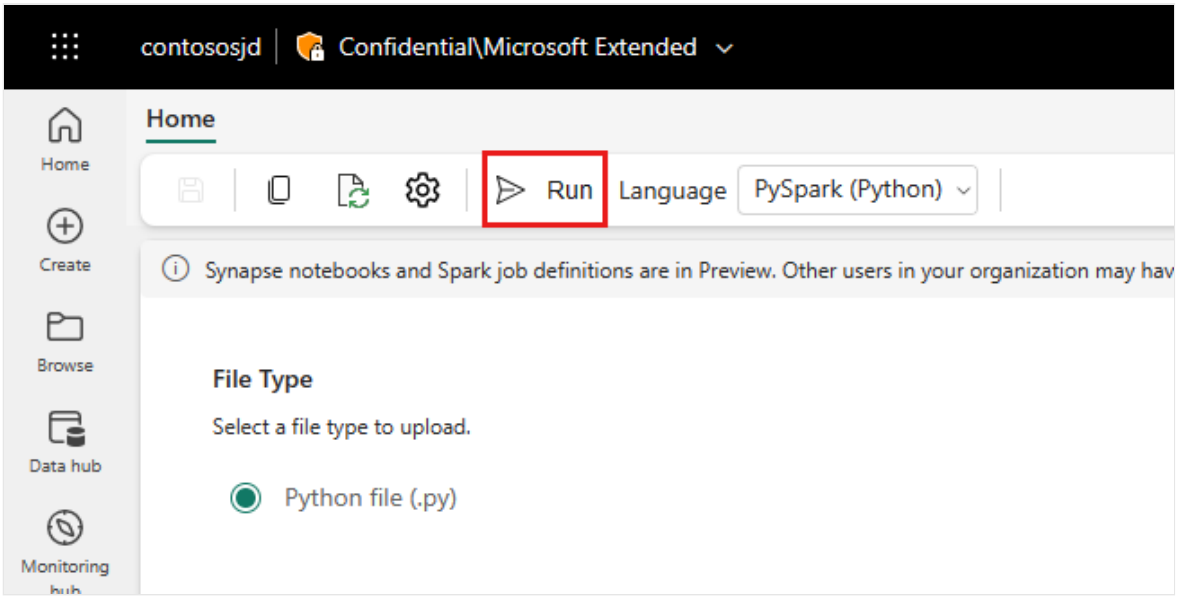


3. Define maximum retry attempts or check **Allow unlimited attempts**.

4. Specify time between each retry attempt and select **Apply**.

Execute and monitor the Spark Job Definition

1. From the top menu, select the **Run** icon.



2. Verify if the **Spark Job definition** was submitted successfully and running.

View data using a SQL endpoint

1. In workspace view, select your Lakehouse.

2. From the right corner, select **Lakehouse** and select **SQL endpoint**.

3. In the SQL endpoint view under **Tables**, select the table that your script uses to land data. You can then preview your data from the SQL endpoint.

Next steps

- Spark Job Definition
- What is SQL Endpoint for a lakehouse?

Referencing data in lakehouse for Data Science projects

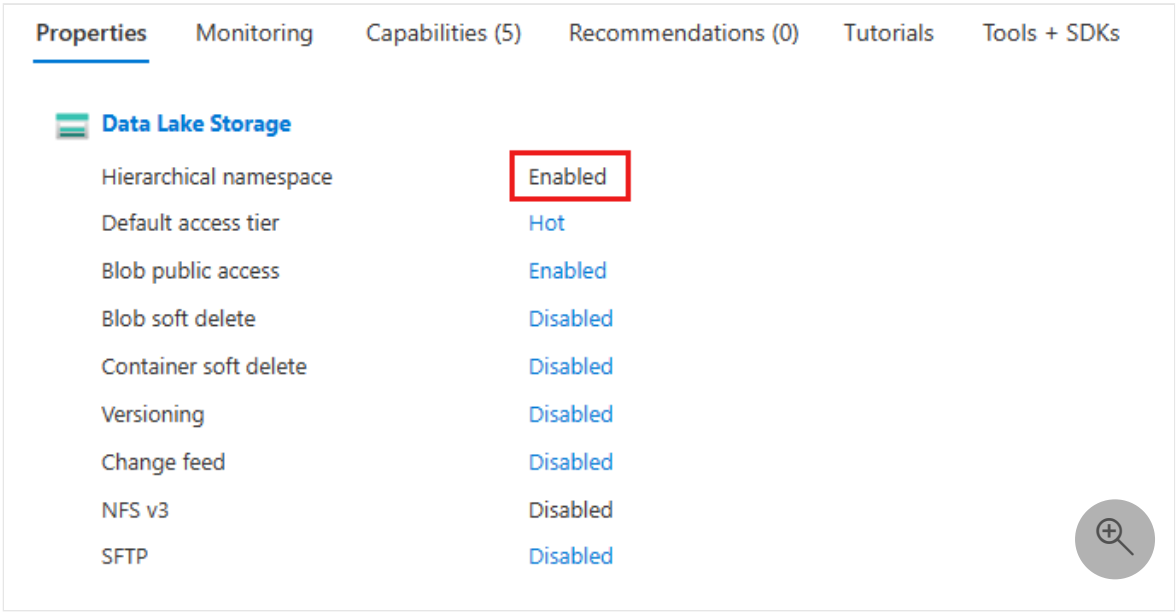
Article • 05/23/2023

This quickstart explains how to reference data stored in external ADLS account and use it in your Data science projects. After completing this quickstart, you'll have a shortcut to ADLS storage in your lakehouse and a notebook with Spark code that accesses your external data.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Prepare data for shortcut

1. In Azure create ADLS Gen2 account
2. Enable hierarchical namespaces



3. Create folders for your data
4. Upload data
5. Add your user identity to BlobStorageContributor role

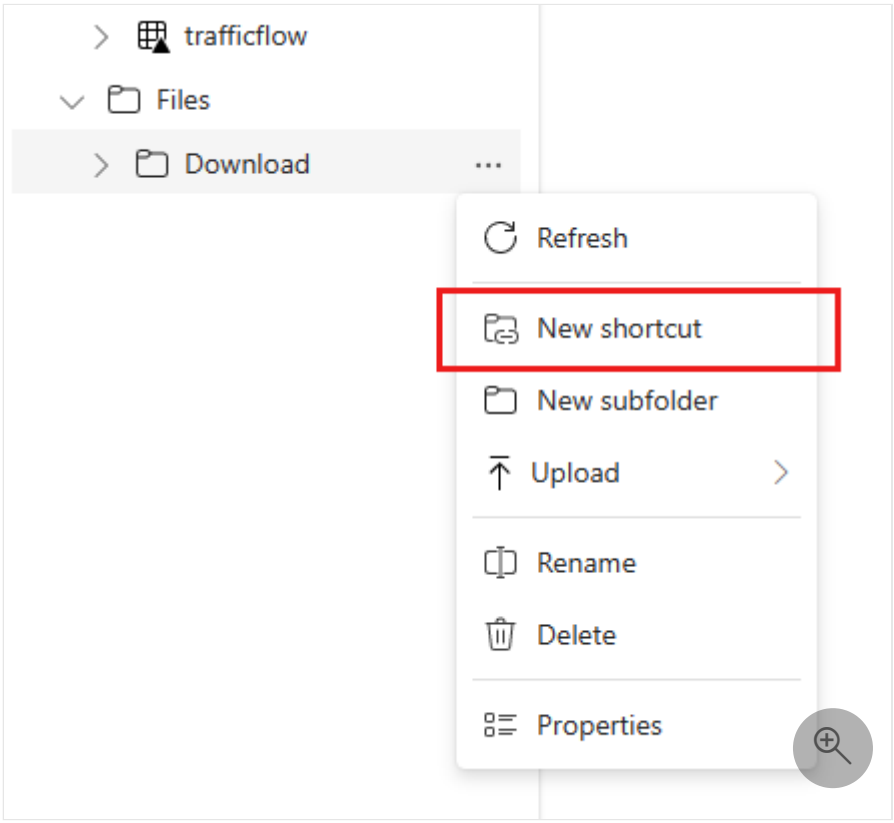
6. Get storage account endpoint

Create a shortcut

1. Open your lakehouse to get to Lakehouse Explorer

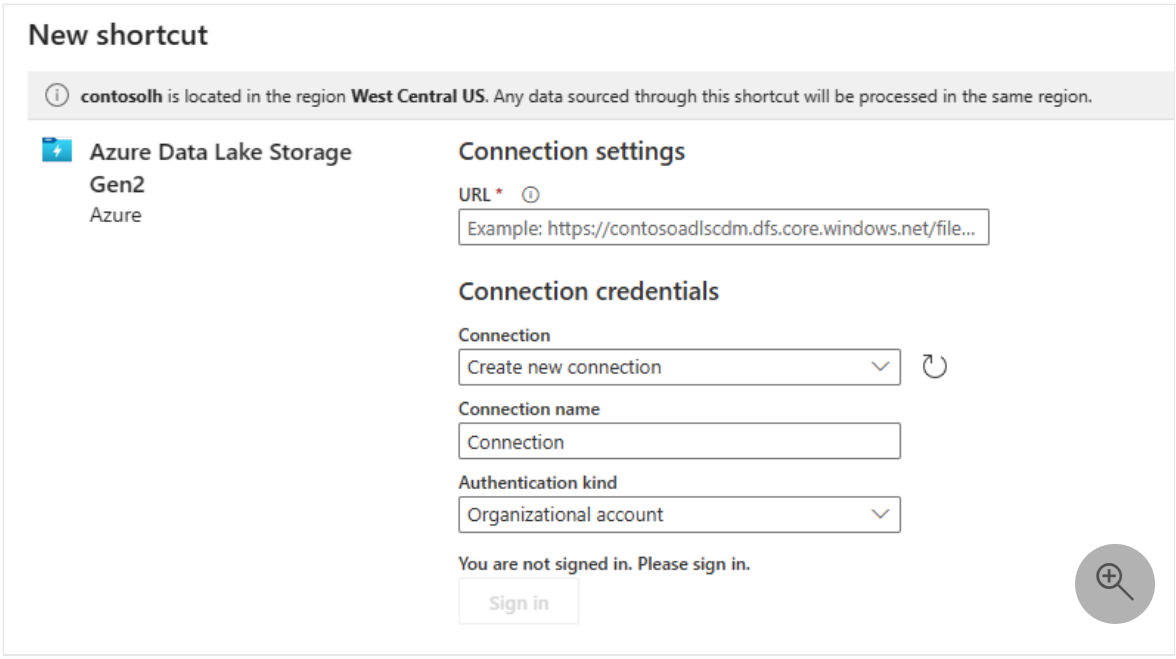
2. Under files create a folder where you reference data

3. Right select (...) and select New Shortcut next to the folder name



4. Select External Sources > ADLS Gen2

5. Provide shortcut name, storage account endpoint, end your data folder location in storage account



6. Select create

Access referenced data in Notebook

1. Open existing or create new notebook
2. Pin your lakehouse to the notebook
3. Browse your data in shortcut folder
4. Select a file with structured data and drag it to notebook to get code generated
5. Execute code to get file content
6. Add code for data analysis

Next steps

- Load to Delta Lake tables
- What is Delta Lake?

Lakehouse end-to-end scenario: overview and architecture

Article • 05/23/2023

Microsoft Fabric is an all-in-one analytics solution for enterprises that covers everything from data movement to data science, real-time analytics, and business intelligence. It offers a comprehensive suite of services, including data lake, data engineering, and data integration, all in one place. For more information, see What is Microsoft Fabric?

This tutorial walks you through an end-to-end scenario from data acquisition to data consumption. It helps you build a basic understanding of Fabric, including the different experiences and how they integrate, as well as the professional and citizen developer experiences that come with working on this platform. This tutorial isn't intended to be a reference architecture, an exhaustive list of features and functionality, or a recommendation of specific best practices.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Lakehouse end-to-end scenario

Traditionally, organizations have been building modern data warehouses for their transactional and structured data analytics needs. And data lakehouses for big data (semi/unstructured) data analytics needs. These two systems ran in parallel, creating silos, data duplicity, and increased total cost of ownership.

Fabric with its unification of data store and standardization on Delta Lake format allows you to eliminate silos, remove data duplicity, and drastically reduce total cost of ownership.

With the flexibility offered by Fabric, you can implement either lakehouse or data warehouse architectures or combine these two together to get the best of both with simple implementation. In this tutorial, you're going to take an example of a retail organization and build its lakehouse from start to finish. It uses the medallion architecture where the bronze layer has the raw data, the silver layer has the validated

and deduplicated data, and the gold layer has highly refined data. You can take the same approach to implement a lakehouse for any organization from any industry.

This tutorial explains how a developer at the fictional Wide World Importers company from the retail domain completes the following steps:

1. Sign in to your Power BI account, or if you don't have one yet, sign up for a free trial.

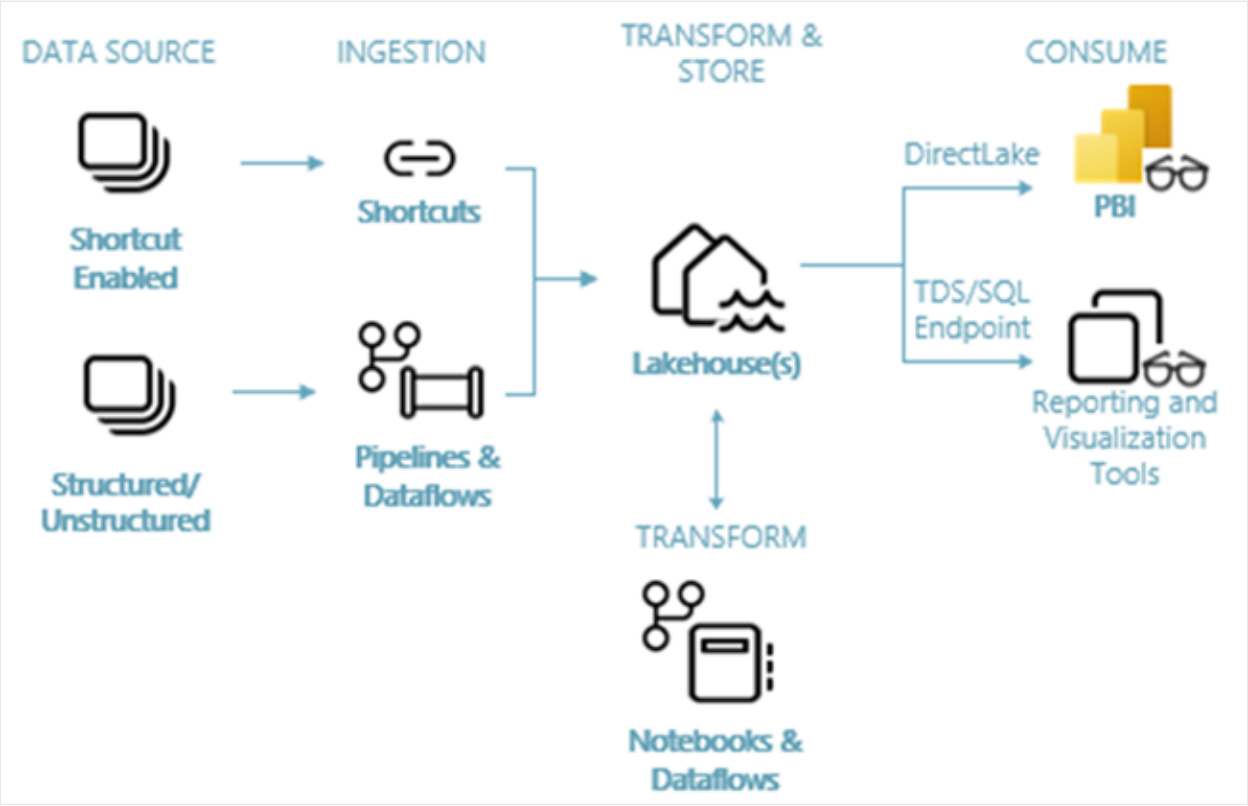
2. Build and implement an end-to-end lakehouse for your organization:

    - Create a Fabric workspace
    - Create a lakehouse. It includes an optional section to implement the medallion architecture that is the bronze, silver, and gold layers.
    - Ingest data, transform data, and load it into the lakehouse. Load data from the bronze, silver, and gold zones as delta lake tables. You can also explore the OneLake, OneCopy of your data across lake mode and warehouse mode.
    - Connect to your lakehouse using TDS/SQL endpoint and Create a Power BI report using DirectLake to analyze sales data across different dimensions.
    - Optionally, you can orchestrate and schedule data ingestion and transformation flow with a pipeline.

3. Clean up resources by deleting the workspace and other items.

Architecture

The following image shows the lakehouse end-to-end architecture. The components involved are described in detailed below:



- **Data sources**: Fabric makes it quick and easy to connect to Azure Data Services, as well as other cloud-based platforms and on-premises data sources, for streamlined data ingestion.
- **Ingestion**: You can quickly build insights for your organization using more than 200 native connectors. These connectors are integrated into the Fabric pipeline and utilize the user-friendly drag-and-drop data transformation with dataflow. Additionally, with the Shortcut feature in Fabric you can connect to existing data, without having to copy or move it.
- **Transform and store**: Fabric standardizes on Delta Lake format. Which means all the Fabric engines can access and manipulate the same dataset stored in OneLake without duplicating data. This storage system provides the flexibility to build lakehouses using a medallion architecture or a data mesh, depending on your organizational requirement. You can choose between a low-code or no-code experience for data transformation, utilizing either pipelines/dataflows or notebook/Spark for a code-first experience.
- **Consume**: Power BI can consume data from the Lakehouse for reporting and visualization. Each Lakehouse has a built-in TDS/SQL endpoint, for easy connectivity and querying of data in the Lakehouse tables from other reporting tools. Additionally, when a Lakehouse is created, a corresponding secondary item called a Warehouse is automatically generated with the same name as the Lakehouse. It provides users with the TDS/SQL endpoint functionality.

Sample dataset

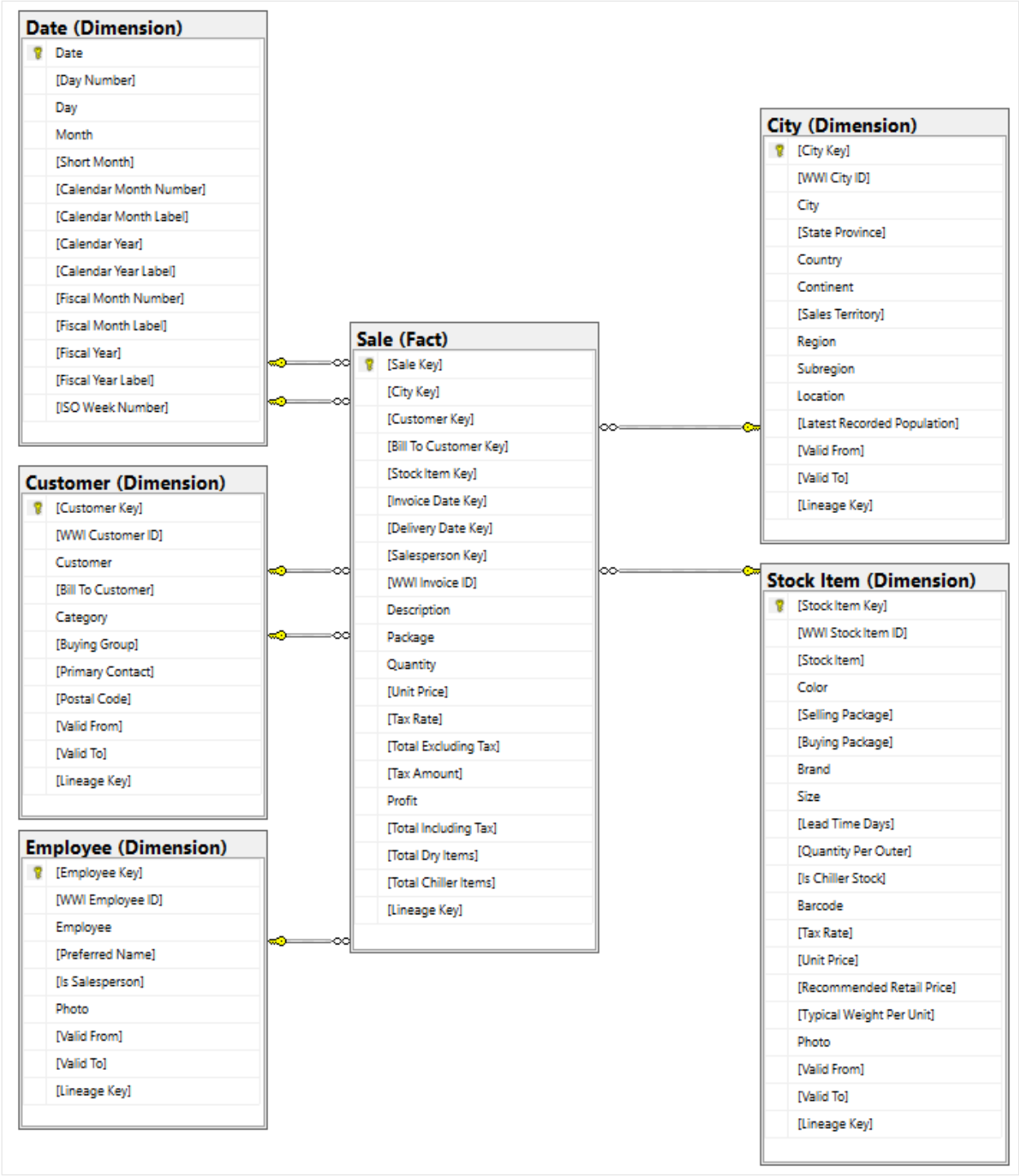
This tutorial uses the Wide World Importers (WWI) sample database. For the lakehouse end-to-end scenario, we have generated sufficient data to explore the scale and performance capabilities of the Fabric platform.

Wide World Importers (WWI) is a wholesale novelty goods importer and distributor operating from the San Francisco Bay area. As a wholesaler, WWI's customers mostly include companies who resell to individuals. WWI sells to retail customers across the United States including specialty stores, supermarkets, computing stores, tourist attraction shops, and some individuals. WWI also sells to other wholesalers via a network of agents who promote the products on WWI's behalf. To learn more about their company profile and operation, see Wide World Importers sample databases for Microsoft SQL.

In general, data is brought from transactional systems or line-of-business applications into a lakehouse. However, for the sake of simplicity in this tutorial, we will use the dimensional model provided by WWI as our initial data source. We use it as the source to ingest the data into a lakehouse and transform it through different stages (Bronze, Silver, and Gold) of a medallion architecture.

Data model

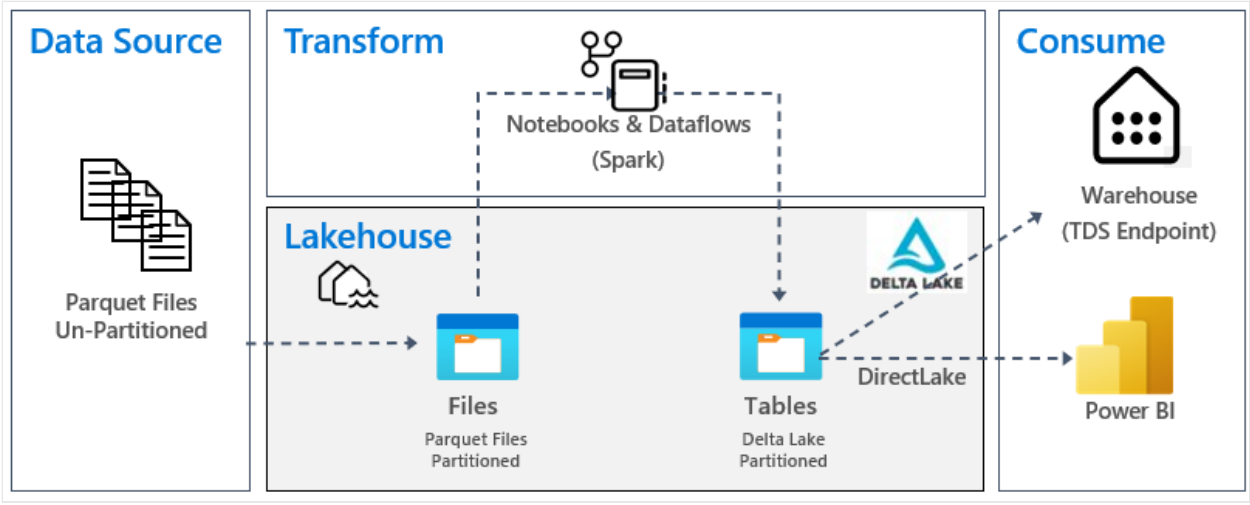
While the WWI dimensional model contains numerous fact tables, for this tutorial, we will use the *Sale* fact table and its correlated dimensions. The following example illustrates the WWI data model:



Data and transformation flow

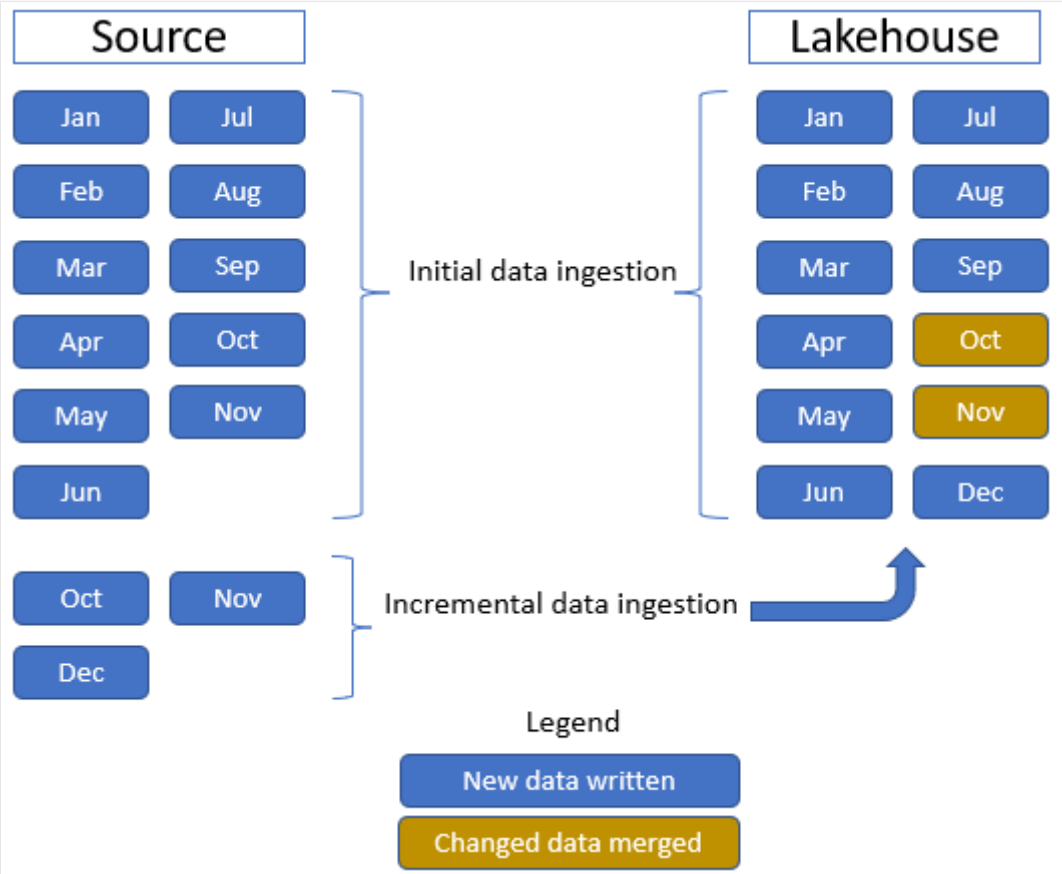
As described earlier, we will use the sample data from Wide World Importers (WWI) sample data to build this end-to-end lakehouse. In this implementation, the sample data is stored in an Azure Data storage account in Parquet file format for all the tables. However, in real-world scenarios, data would typically originate from various sources and in diverse formats.

The following image shows the source, destination and data transformation:



- **Data Source**: The source data is in Parquet file format and in an unpartitioned structure. It's stored in a folder for each table. In this tutorial, we set up a pipeline to ingest the complete historical or onetime data to the lakehouse.

  To demonstrate the capabilities for incremental data load, we have an optional tutorial at the end of this tutorial. In that tutorial, we use the *Sale* fact table, which has one parent folder with historical data for 11 months (with one subfolder for each month) and another folder containing incremental data for three months (one subfolder for each month). During the initial data ingestion, 11 months of data are ingested into the lakehouse table. However, when the incremental data arrives, it includes updated data for Oct and Nov, and new data for Dec. Oct and Nov data is merged with the existing data and the new Dec data is written into lakehouse table as shown in the following image:



- **Lakehouse**: In this tutorial, you will create a lakehouse, ingest data into the files section of the lakehouse and then create delta lake tables in the Tables section of the lakehouse. You can find an optional tutorial, which covers creating the lakehouse with medallion architecture some recommendations.
- **Transform**: For data preparation and transformation, you will see two different approaches. We will demonstrate the use of Notebooks/Spark for users who prefer a code-first experience and use pipelines/dataflow for users who prefer a low-code or no-code experience.
- **Consume**: To demonstrate data consumption, you will see how you can use the DirectLake feature of Power BI to create reports, dashboards and directly query data from the lakehouse. Additionally, we will demonstrate how you can make your data available to third party reporting tools by using the TDS/SQL endpoint. This endpoint allows you to connect to the warehouse and run SQL queries for analytics.

Next steps

Advance to the next article to learn how to

Create a lakehouse

Lakehouse tutorial: Create a Fabric workspace

Article • 05/23/2023

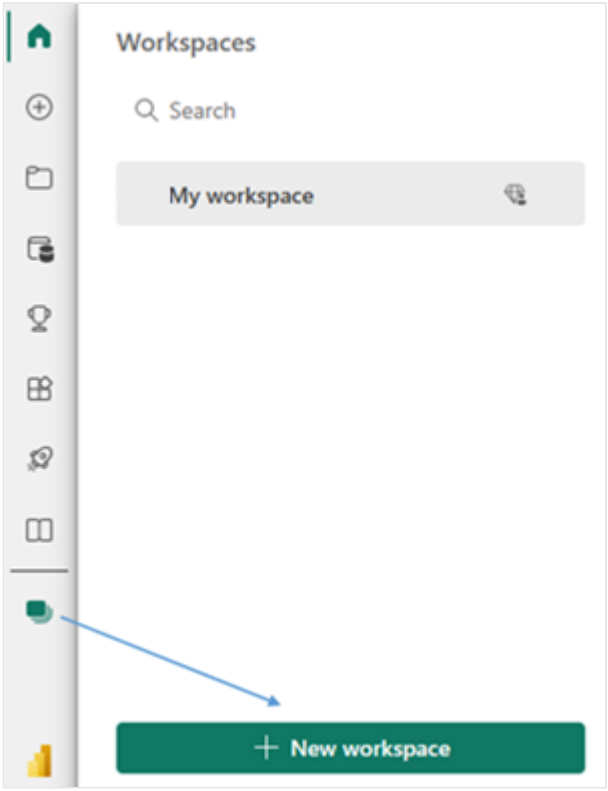
Before you can begin creating the lakehouse, you need to create a workspace where you'll build out the remainder of the tutorial.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Create a workspace

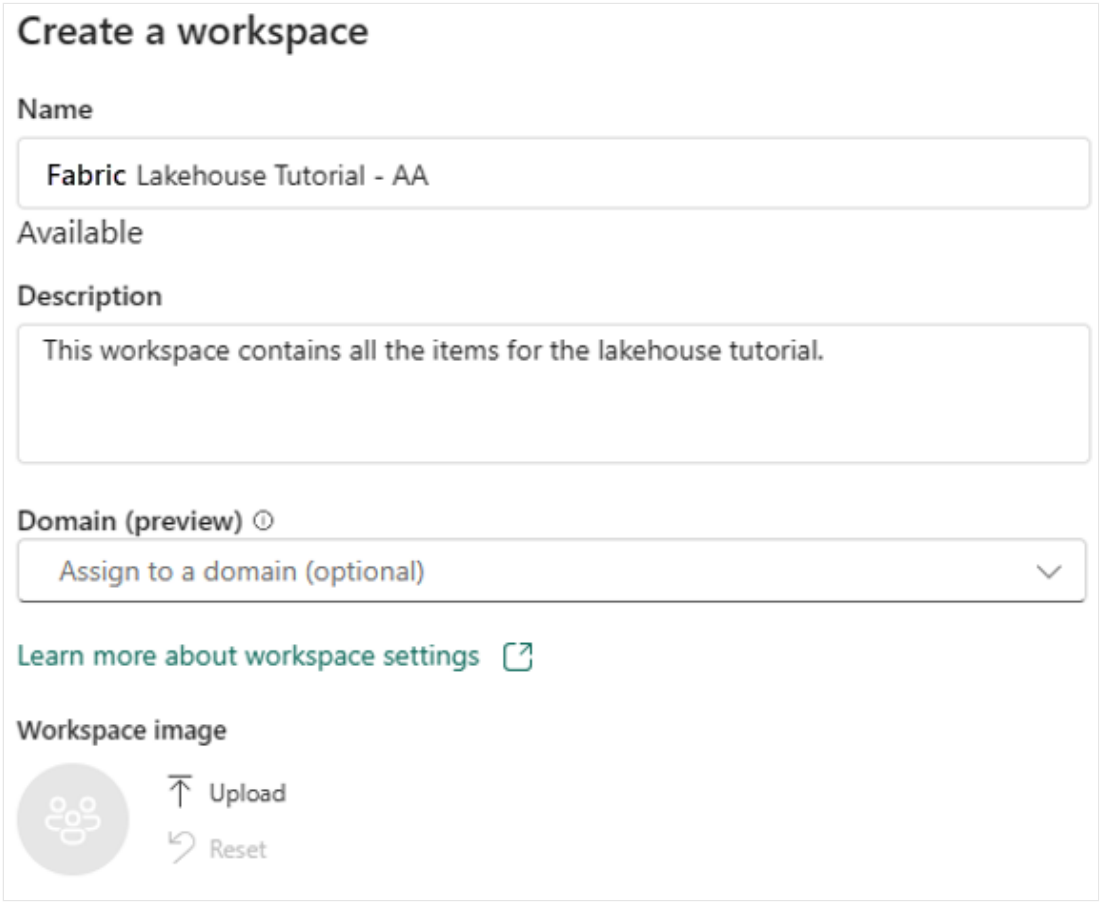
In this step, you create a Fabric workspace. The workspace contains all the items needed for this lakehouse tutorial, which includes lakehouse, dataflows, Data Factory pipelines, the notebooks, Power BI datasets, and reports.

1. Sign in to Power BI.
2. Select **Workspaces** and **New workspace**.

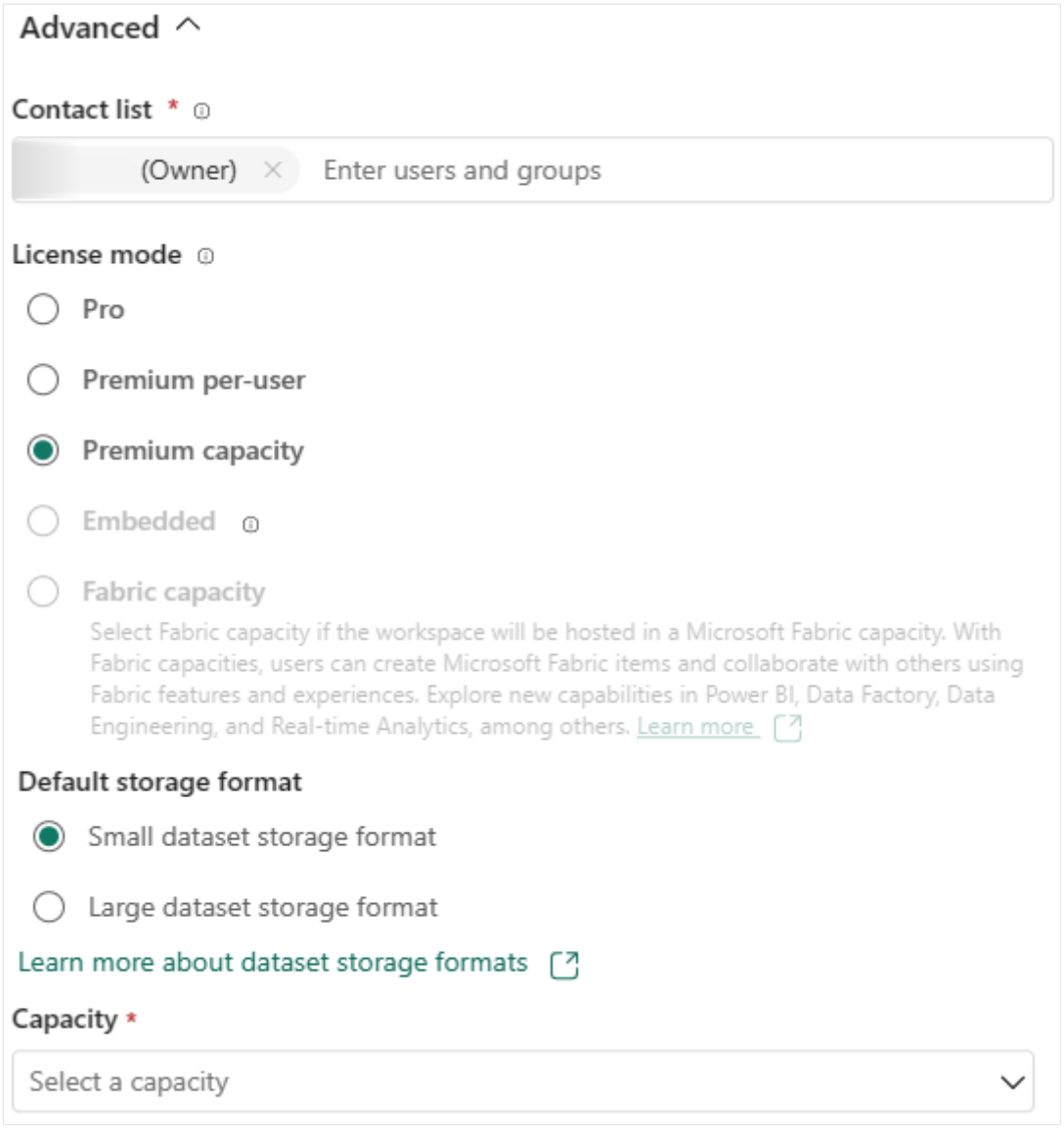


3. Fill out the **Create a workspace** form with the following details:

   - **Name:** Enter *Fabric Lakehouse Tutorial*, and any extra characters to make the name unique.
   - **Description**: Enter an optional description for your workspace.



   - **Advanced**: Under **License mode**, select **Premium capacity** and then choose a premium capacity that you have access to.



4. Select **Apply** to create and open the workspace.

Next steps

Advance to the next article to learn about

Create a lakehouse in Microsoft Fabric

Lakehouse tutorial: Create a lakehouse, ingest sample data, and build a report

Article • 05/23/2023

In this tutorial, you'll build a lakehouse, ingest sample data into the delta table, apply transformation where required, and then create reports.

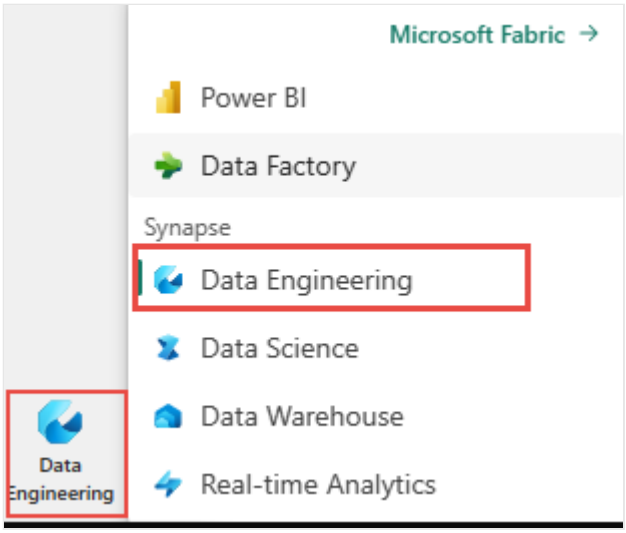
> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Prerequisites

- Create a Fabric workspace
- In this article, you'll follow steps to ingest a CSV file, which requires you to have OneDrive configured. If you don't have OneDrive configured, sign up for the Microsoft 365 free trial: Free Trial - Try Microsoft 365 for a month.

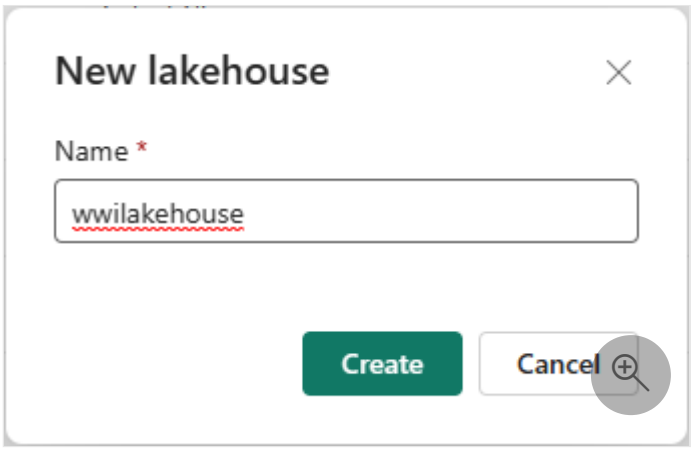
Create a lakehouse

1. In the Power BI service, select **Workspaces** from the left-hand menu.
2. To open your workspace, enter its name in the search textbox located at the top and select it from the search results.
3. From the experience switcher located at the bottom left, select **Data Engineering**.



4. In the **Data Engineering** tab, select **Lakehouse** to create a lakehouse.

5. In the **New lakehouse** dialog box, enter **wwilakehouse** in the **Name** field.



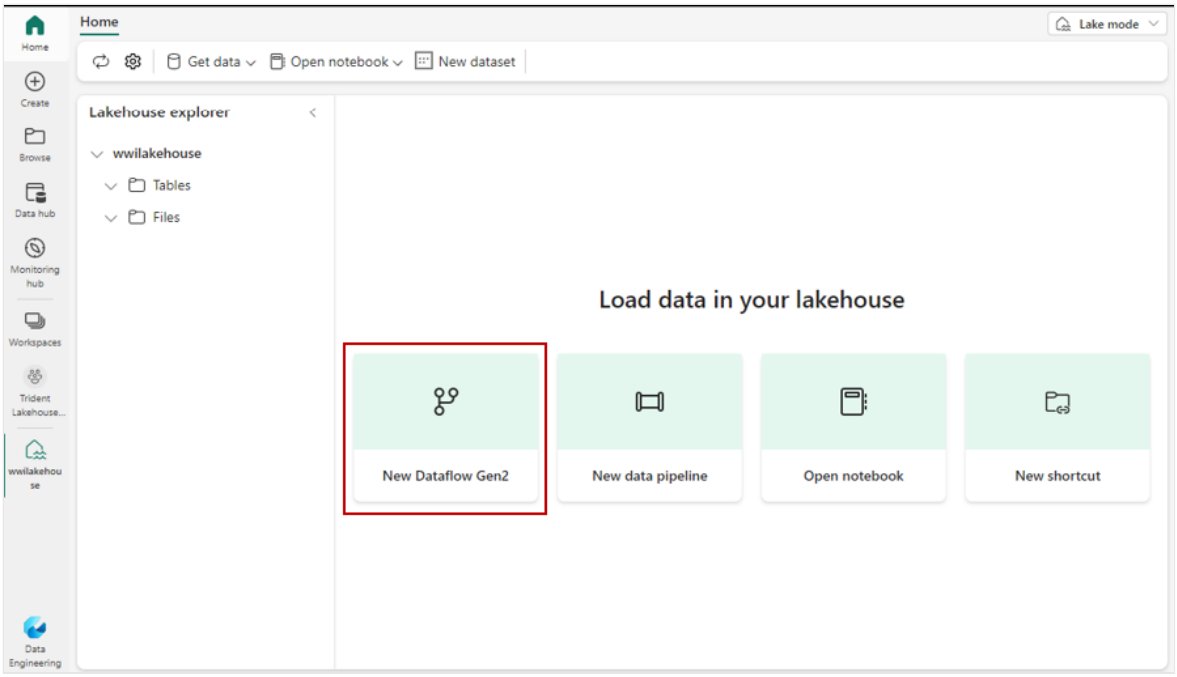
6. Select **Create** to create and open the new lakehouse.

Ingest sample data

> **Note**
>
> If you don't have OneDrive configured, sign up for the Microsoft 365 free trial: **Free Trial - Try Microsoft 365 for a month**.

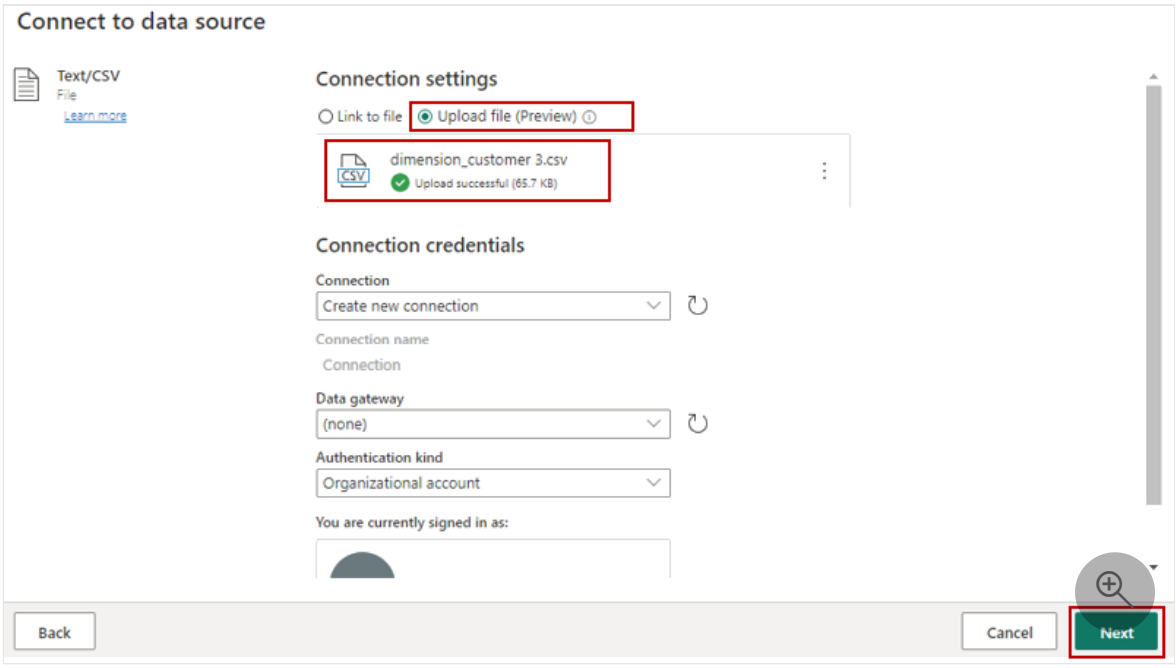
1. Download the *dimension_customer.csv* file from the Fabric samples repo.

2. In the **Lakehouse explorer**, you see options to load data into lakehouse. Select **New Dataflow Gen2**.



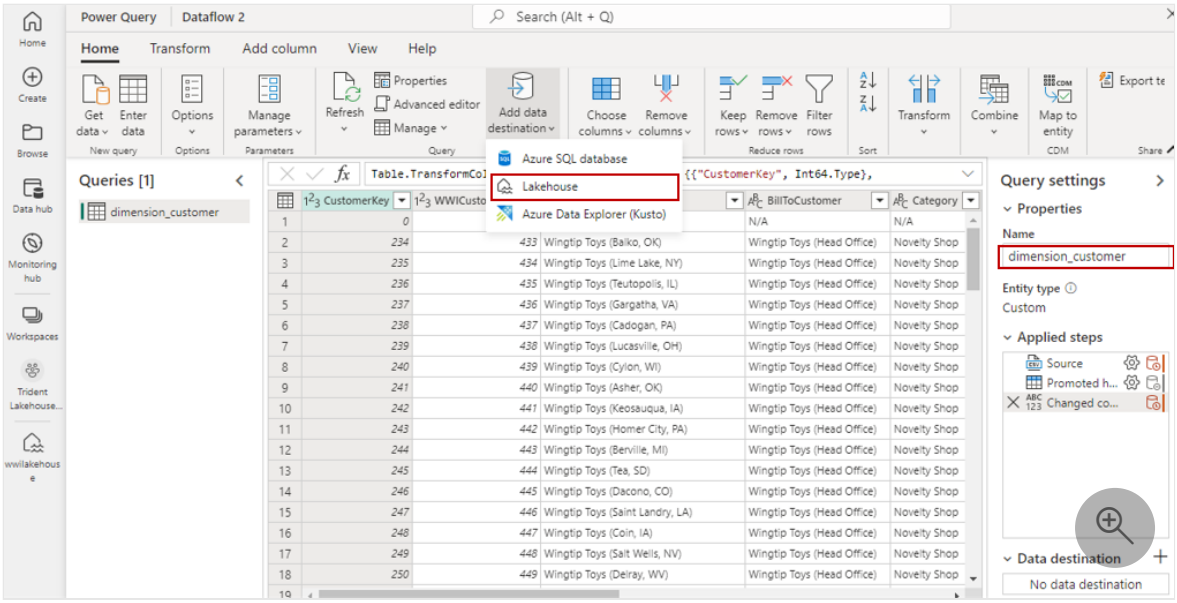
3. On the new dataflow pane, select **Import from a Text/CSV file**.

4. On the **Connect to data source** pane, select the **Upload file** radio button. Drag and drop the *dimension_customer.csv* file that you downloaded in step 1. After the file is uploaded, select **Next**.



5. From the **Preview file data** page, preview the data and select **Create** to proceed and return back to the dataflow canvas.

6. In the **Query settings** pane, enter **dimension_customer** for the **Name** field. From the menu items, select **Add data destination** and select **Lakehouse**.

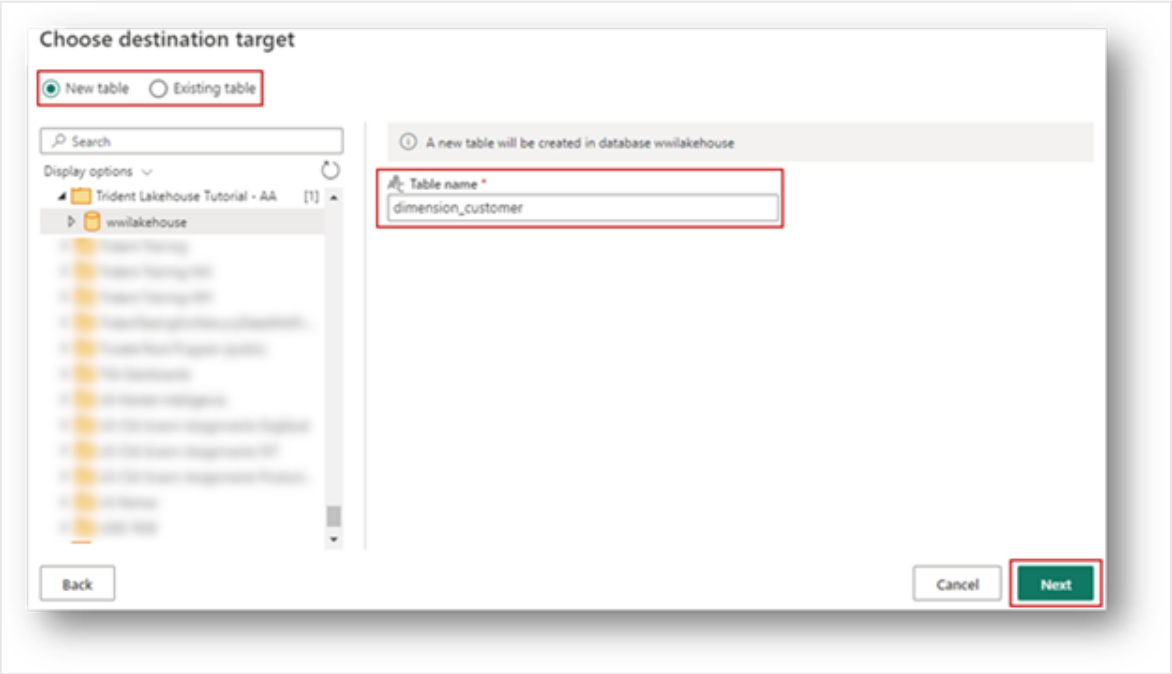


7. If needed, from the **Connect to data destination** screen, sign into your account. Select **Next**.

8. Navigate to the **wwilakehouse** in your workspace.

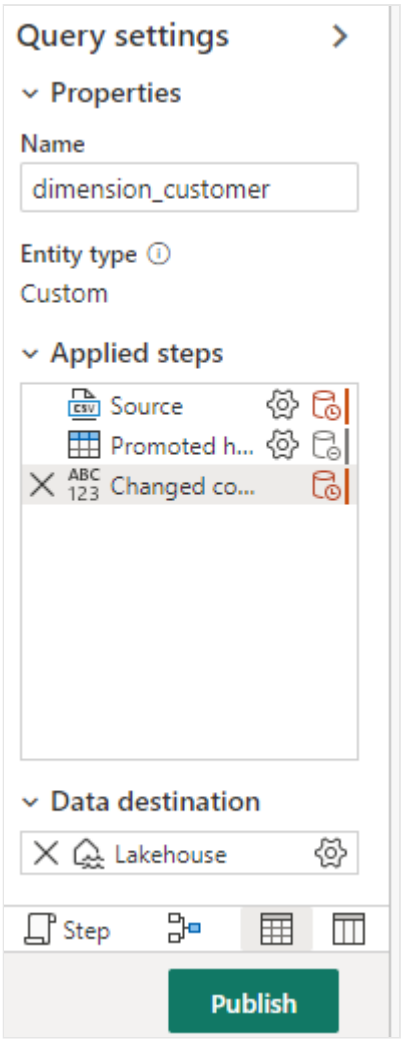
9. If the **dimension_customer** table doesn't exist, select the **New table** setting and enter the table name **dimension_customer**. If the table already exists, select the **Existing table** setting and choose **dimension_customer** from the list of tables in the object explorer. Select **Next**.

> **Note**
>
> Fabric adds a space and number at the end of the table name by default. Table names must be lower case and must not contain spaces. Please rename it appropriately and remove any spaces from the table name.



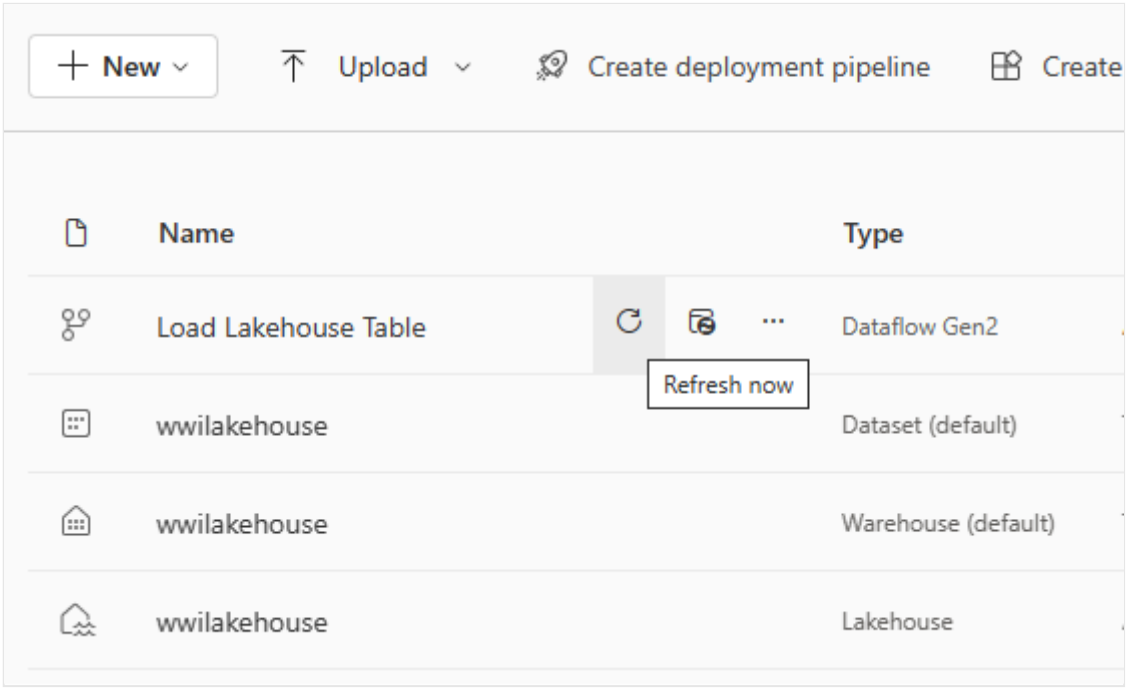
10. On the **Choose destination settings** pane, select **Replace** as **Update method**. Select **Save Settings** to return to the dataflow canvas.

11. From the dataflow canvas, you can easily transform the data based on your business requirements. For simplicity, we aren't making any changes in this tutorial. To proceed, select **Publish** at the bottom right of the screen.

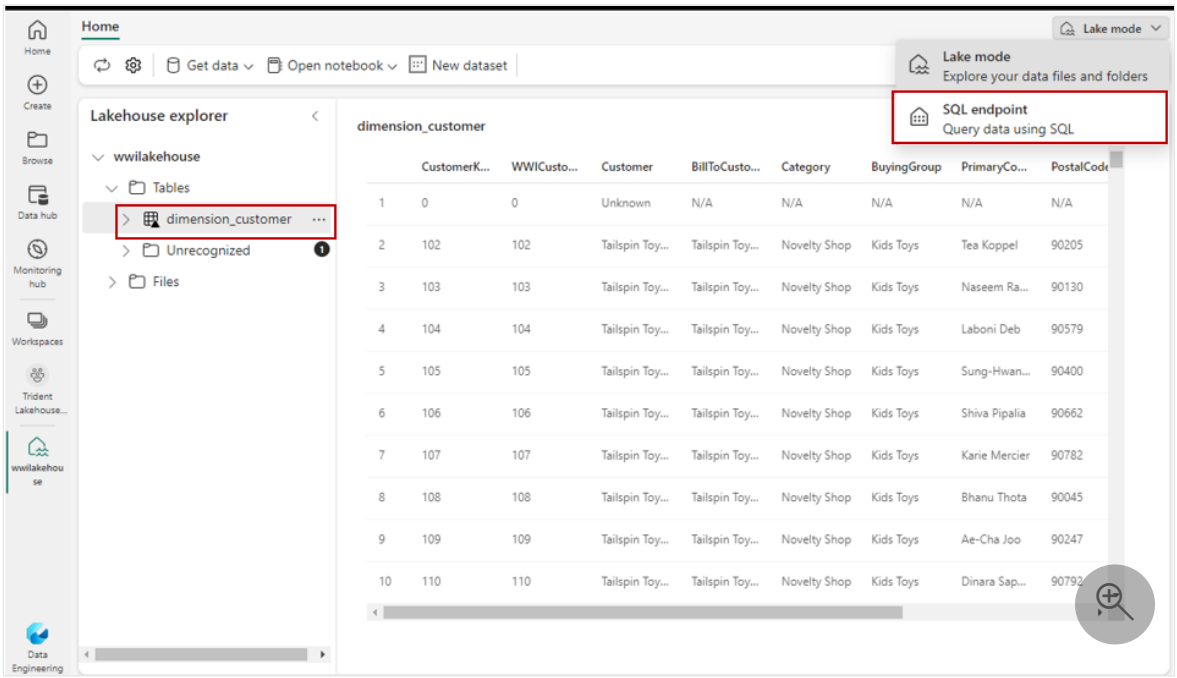


12. A spinning circle next to the dataflow's name indicates publishing is in progress in the item view. When publishing is complete, select the ... and select **Properties**. Rename the dataflow to **Load Lakehouse Table** and select **Save**.

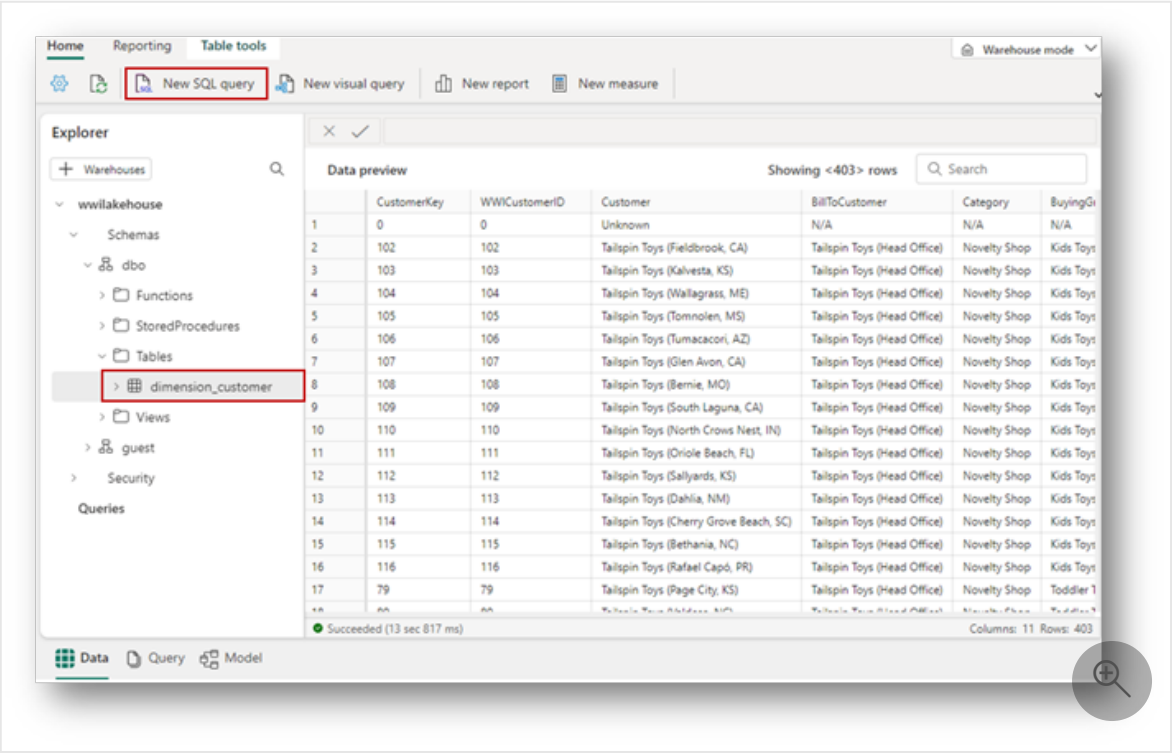
13. Select the **Refresh now** option next to data flow name to refresh the dataflow. It runs the dataflow and moves data from the source file to lakehouse table. While it's in progress, you see a spinning circle under **Refreshed** column in the item view.



14. Once the dataflow is refreshed, go to the lakehouse to view the **dimension_customer** delta table. Select it to preview its data. You can also use the SQL endpoint of the lakehouse to query the data with SQL statements in the warehouse mode. Select **SQL endpoint** under **Lake mode** at the top right of the screen.



15. In warehouse mode, select the **dimension_customer** table to preview its data or select **New SQL query** to write your SQL statements.



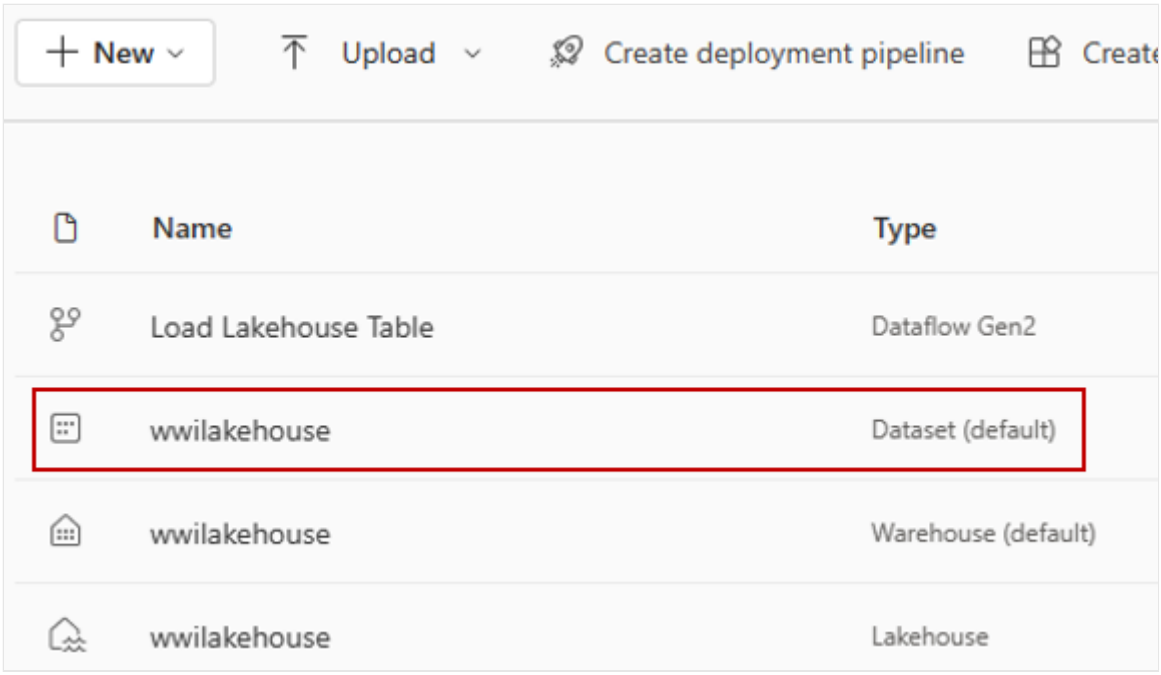
16. The following sample query aggregates the row count based on the *BuyingGroup* column of the *dimension_customer* table. SQL query files are saved automatically for future reference, and you can rename or delete these files based on your need.

    To run the script, select the **Run** icon at the top of the script file.

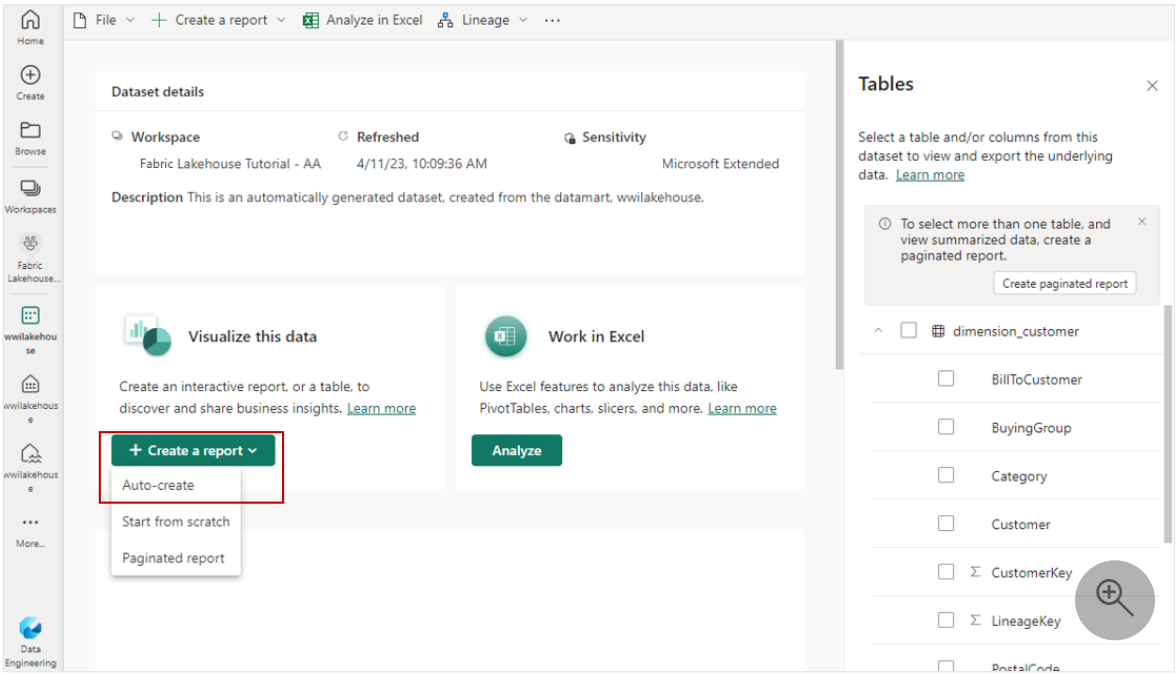
```SQL
SELECT BuyingGroup, Count(*) AS Total
FROM dimension_customer
GROUP BY BuyingGroup
```

Build a report

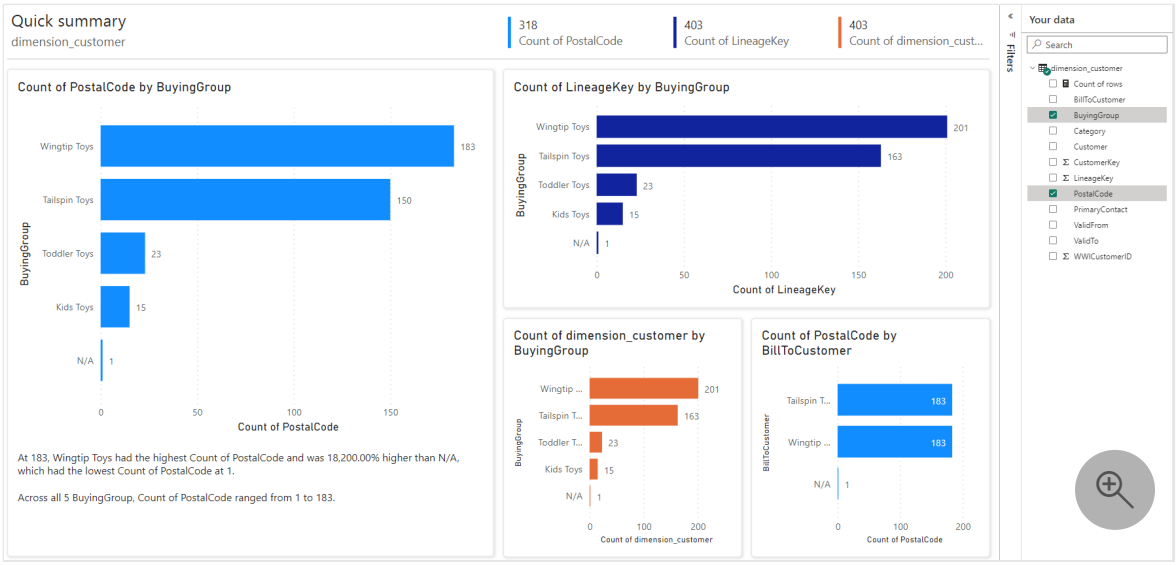
1. In the item view of the workspace, select the **wwilakehouse** default dataset. This dataset is automatically created and has the same name as the lakehouse.



2. From the dataset pane, you can view all the tables. You have options to create reports either from scratch, paginated report, or let Power BI automatically create a report based on your data. For this tutorial, select **Auto-create** under **Create a report**. In the next tutorial, we create a report from scratch.



3. Since the table is a dimension and there are no measures in it, Power BI creates a measure for the row count and aggregates it across different columns, and creates different charts as shown in the following image. You can save this report for the future by selecting **Save** from the top ribbon. You can make more changes to this report to meet your requirement by including or excluding other tables or columns.



Next steps

Advance to the next article to learn how to

Ingest data into the lakehouse

Lakehouse tutorial: Ingest data into the lakehouse

Article • 05/23/2023

In this tutorial, you'll ingest additional dimensional and fact tables from the Wide World Importers (WWI) into the lakehouse.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

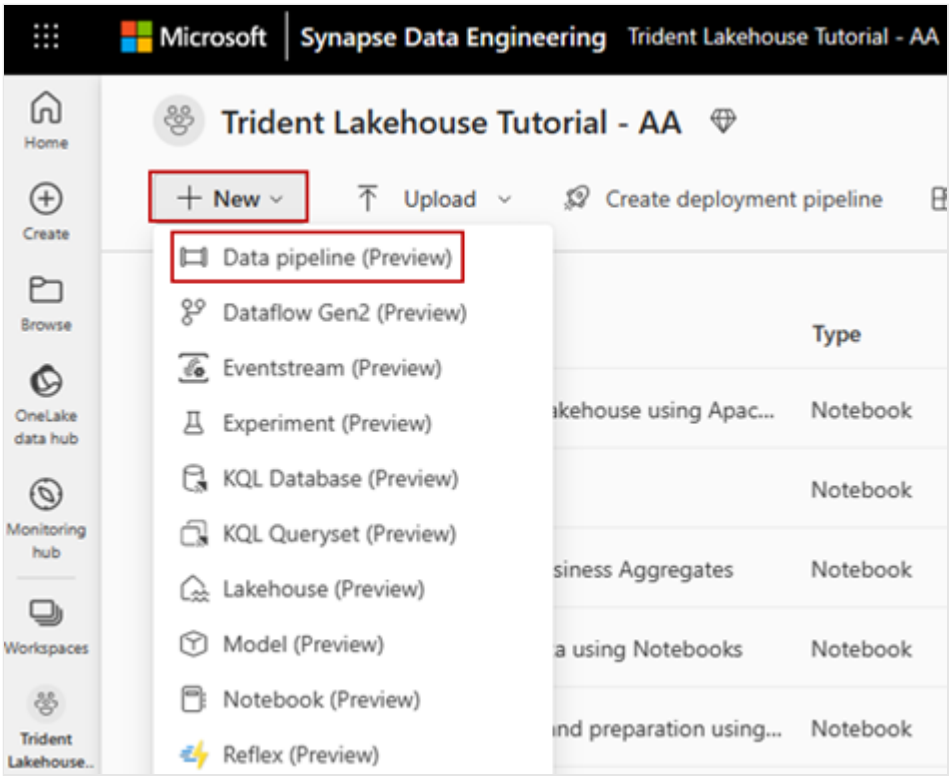
Prerequisites

- Create a lakehouse

Ingest data

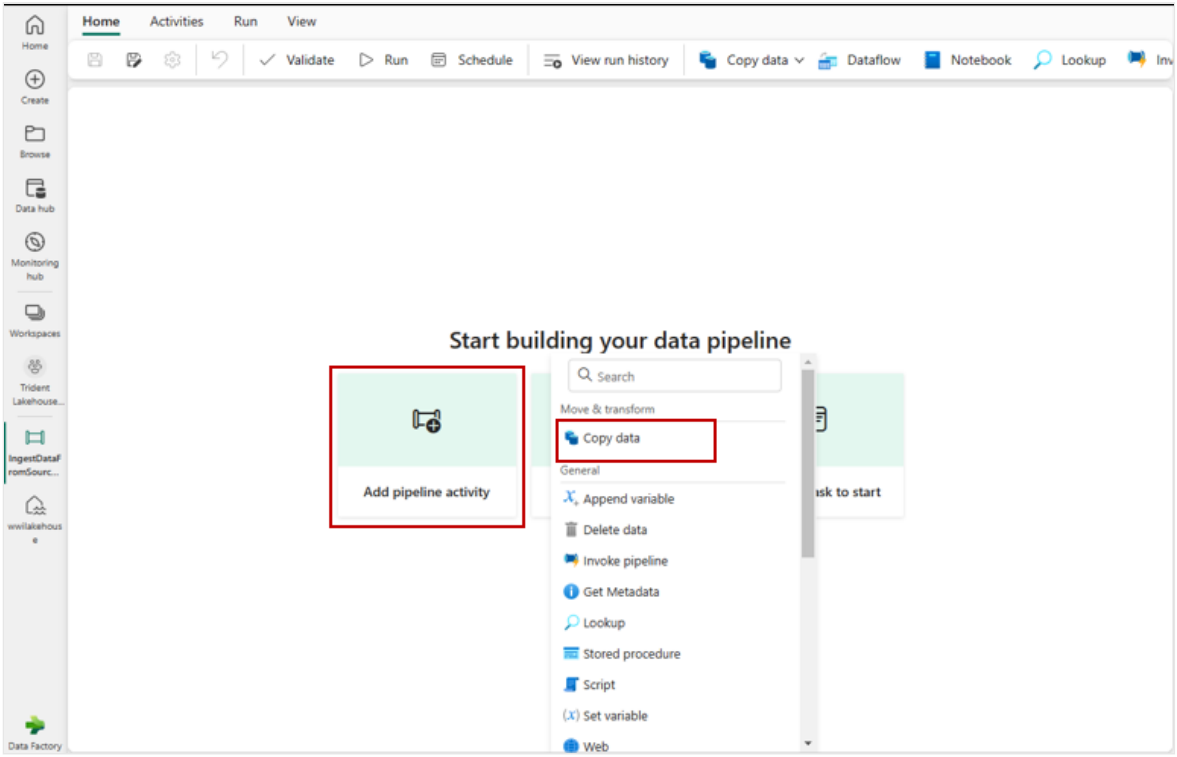
In this section, you use the **Copy data activity** of the Data Factory pipeline to ingest sample data from an Azure storage account to the **Files** section of the lakehouse you created earlier.

1. Choose the workspace you created from the Workspace flyout on left hand. From the **+New** button in the workspace page, select **Data pipeline**



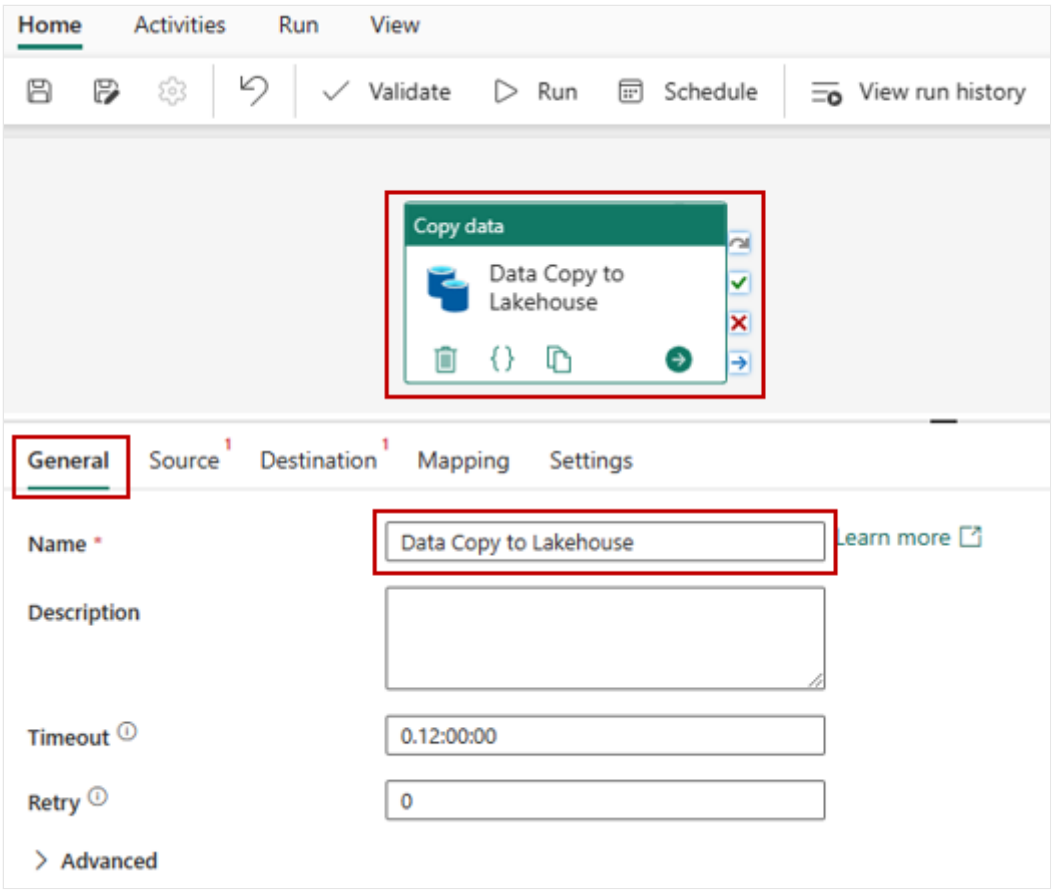
2. In the **New pipeline** dialog box, specify the name as **IngestDataFromSourceToLakehouse** and select **Create**. A new data factory pipeline is created and opened.

3. On your newly created data factory pipeline, select **Add pipeline activity** to add an activity to the pipeline and select **Copy data**. This adds copy data activity to the pipeline canvas.

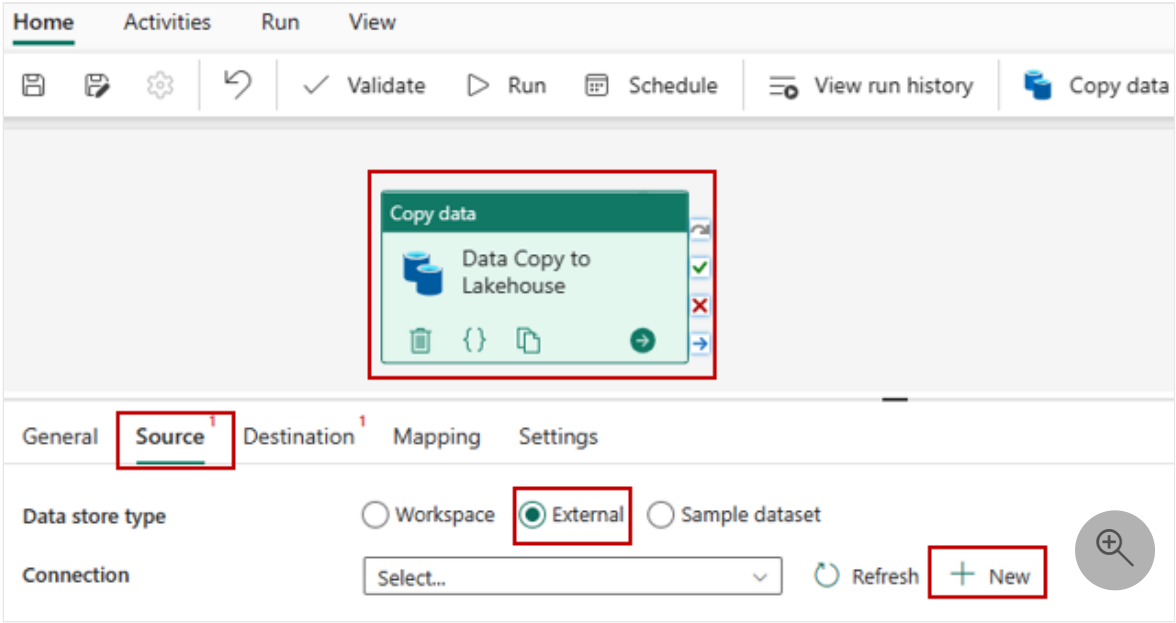


4. Select the newly added copy data activity from the canvas. It shows activity properties at the bottom. Under the **General** tab, specify the name for the copy

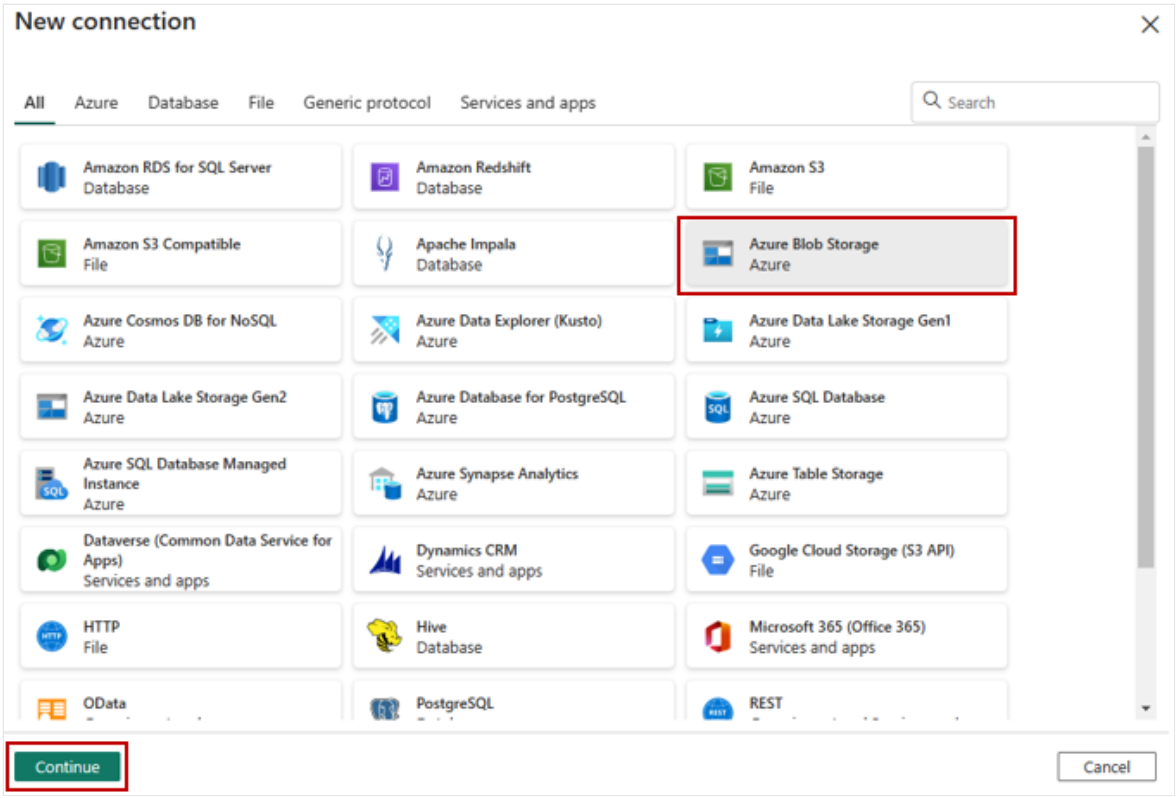
data activity **Data Copy to Lakehouse**.



5. Under **Source** tab of the selected copy data activity, select **External** as **Data store type** and then select **+ New** to create a new connection to data source.

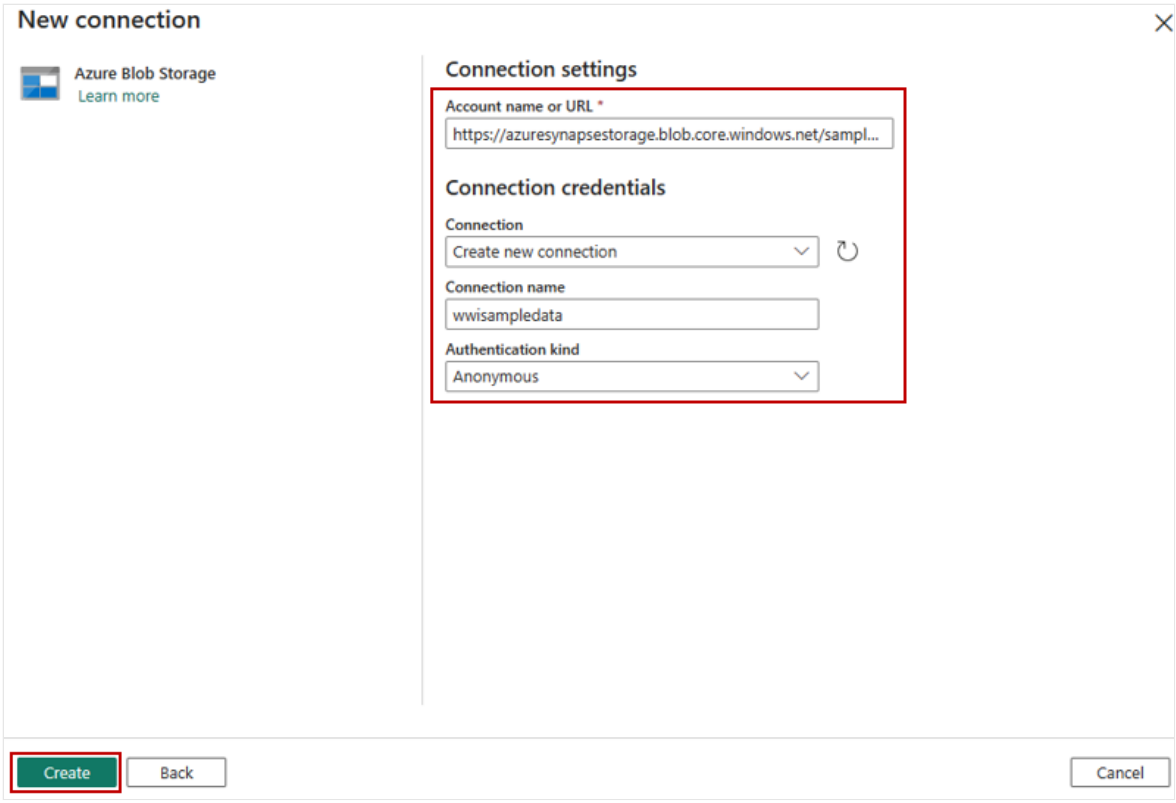


6. For this tutorial, all the sample data is available in a public container of Azure blob storage. You connect to this container to copy data from it. On the **New connection** wizard, select **Azure Blob Storage** and then select **Continue**.



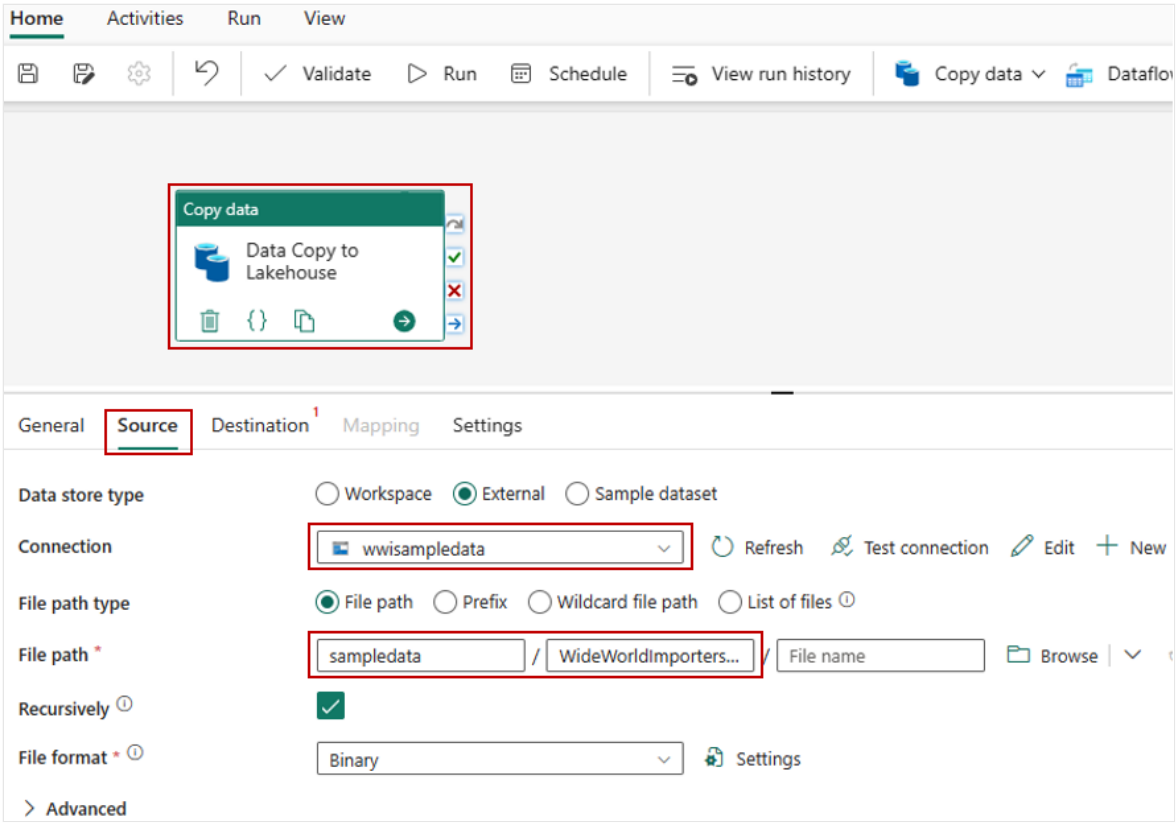
7. On the next screen of the **New connection** wizard, enter the following details and select **Create** to create the connection to the data source.

Property	Value
Account name or URI	`https://azuresynapsestorage.blob.core.windows.net/sampledata`
Connection	Create new connection
Connection name	wwisampledata
Authentication kind	Anonymous



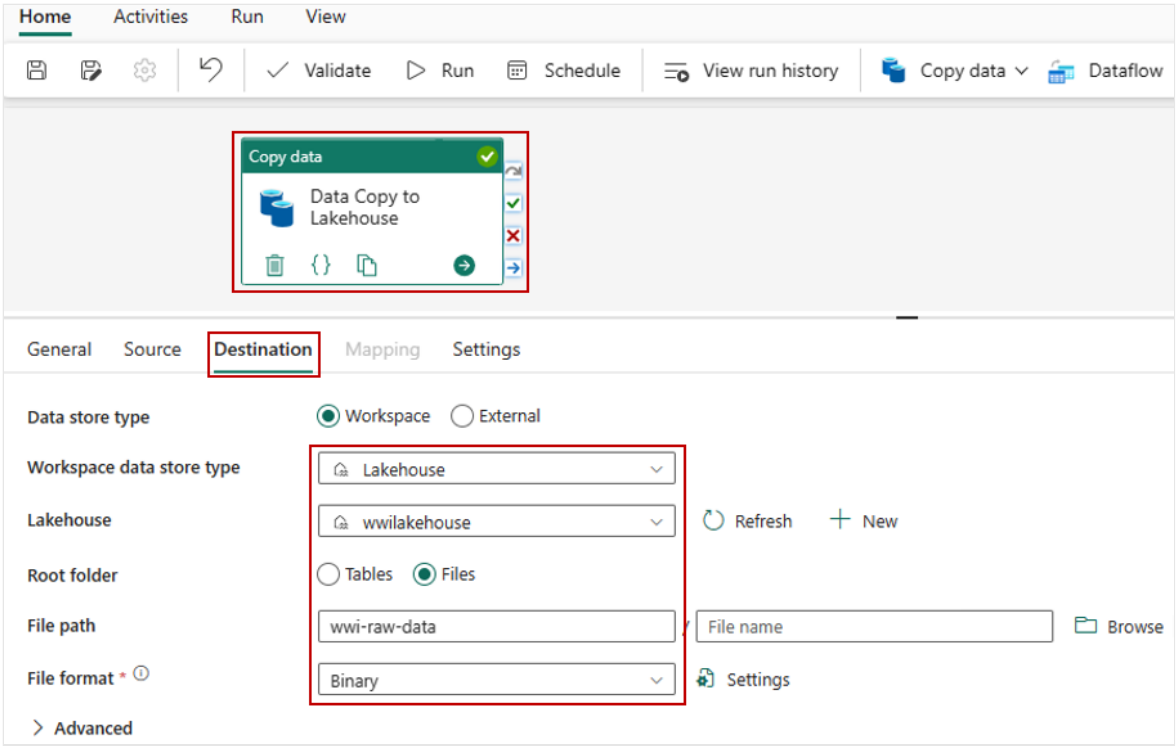
8. Once the new connection is created, return to the **Source** tab of the copy data activity, and the newly created connection is selected by default. Specify the following properties before moving to the destination settings.

Property	**Value**
Data store type	External
Connection	wwisampledata
File path type	File path
File path	Container name (first text box): sampledata Directory name (second text box): WideWorldImportersDW/parquet
Recursively	Checked
File Format	Binary



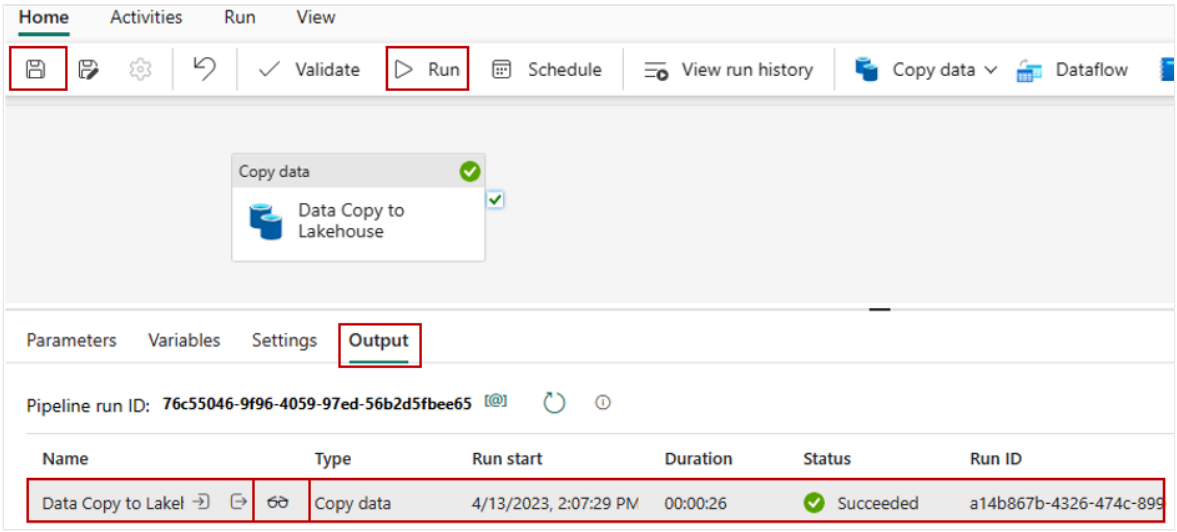
9. Under the **Destination** tab of the selected copy data activity, specify the following properties:

Property	Value
Data store type	Workspace
Workspace data store type	Lakehouse
Lakehouse	wwilakehouse
Root Folder	Files
File path	Directory name (first text box): wwi-raw-data
File Format	Binary

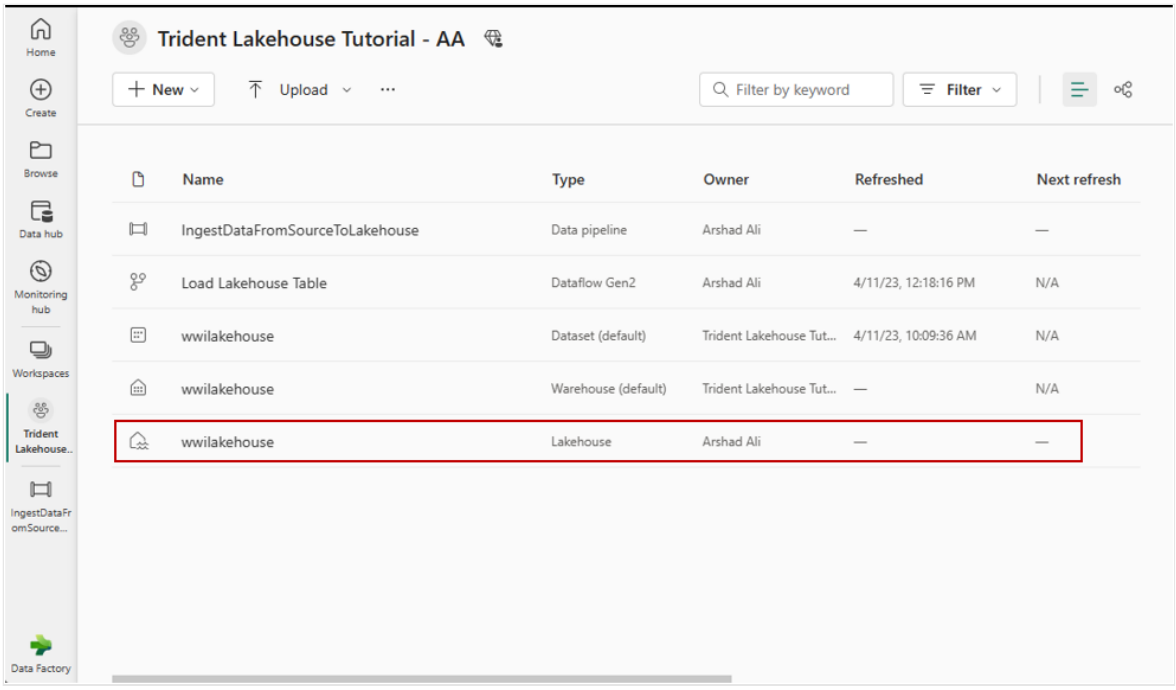


10. You now finished configuring the copy data activity. Select the **Save** button under **Home** to save the changes made, and select **Run** to execute your pipeline and its activity. You can also schedule pipelines to refresh data at defined intervals to meet your business requirements. For this tutorial, we'll run the pipeline only once by clicking on **Run** button.

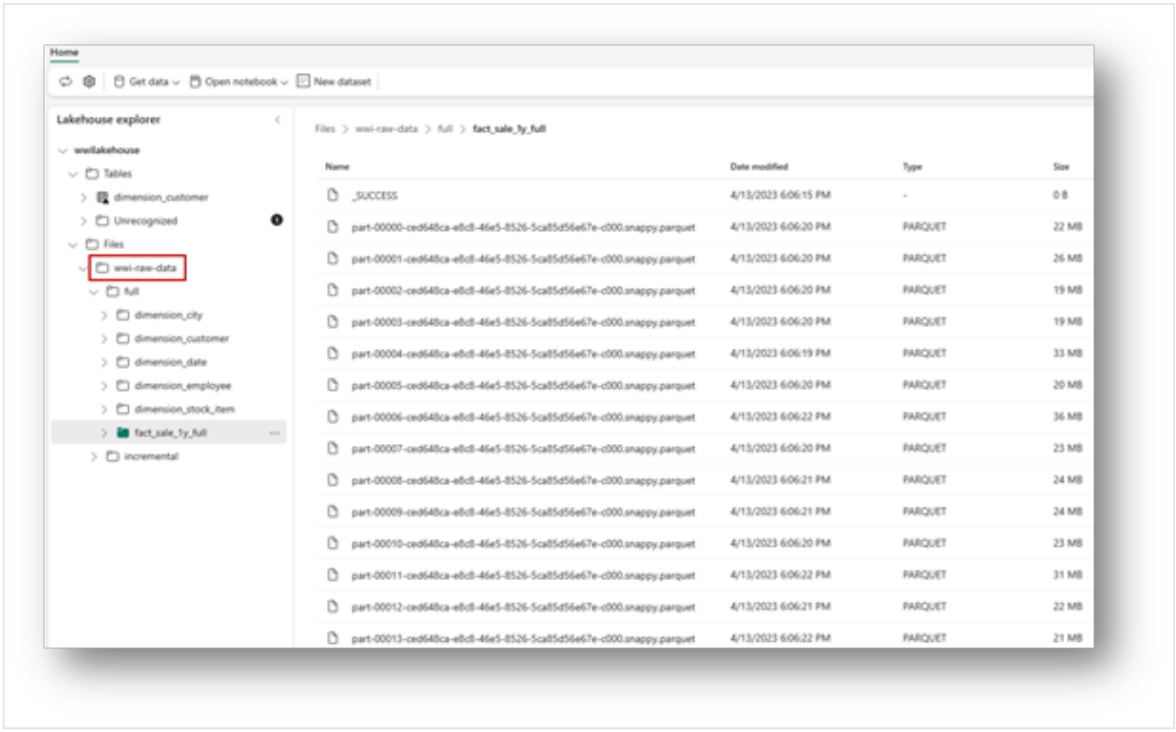
    This triggers data copy from the underlying data source to the specified lakehouse and might take up to a minute to complete. You can monitor the execution of the pipeline and its activity under the **Output** tab, which appears when you click anywhere on the canvas. Optionally, you can select the glasses icon to look at the details of the data transfer.



11. Once the data is copied, go to the items view of the workspace and select **wwilakehouse** to launch the **Lakehouse explorer** for this selected lakehouse.



12. Validate that in the **Lakehouse explorer** view, a new folder **wwi-raw-data** has been created and data for all the tables have been copied there.



Next steps

Advance to the next article to learn about

Prepare and transform data

Lakehouse tutorial: Prepare and transform data in the lakehouse

Article • 05/23/2023

In this tutorial, you use notebooks with Spark runtime to transform and prepare the data.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

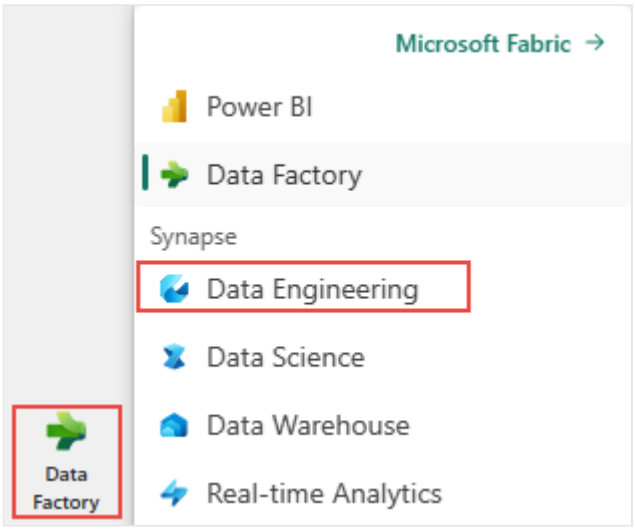
Prerequisites

- Ingest data into the lakehouse

Prepare data

From the previous tutorial steps, we have raw data ingested from the source to the **Files** section of the lakehouse. Now you can transform that data and prepare it for creating delta tables.

1. Download the notebooks from the Lakehouse Tutorial Source Code folder.
2. From the experience switcher located at the bottom left of the screen, select **Data engineering**.

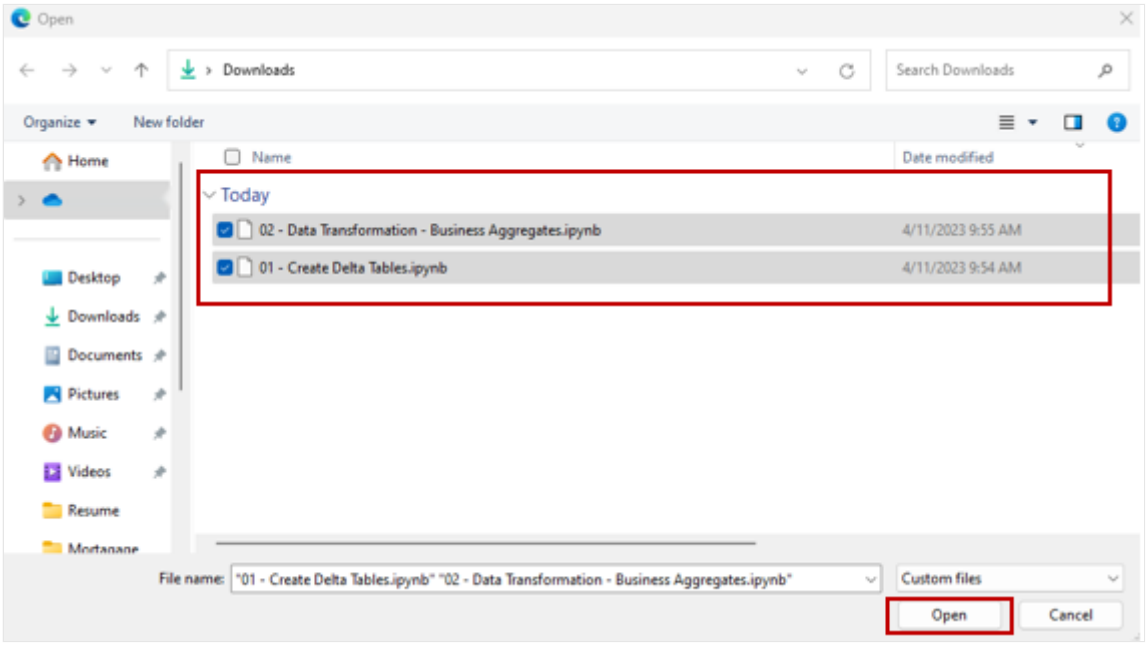


3. Select **Import notebook** from the **New** section at the top of the landing page.

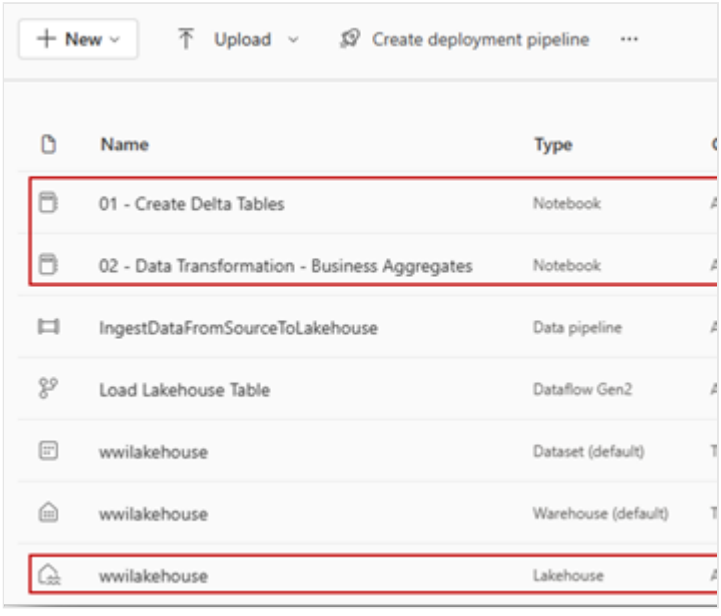
4. Select **Upload** from the **Import status** pane that opens on the right side of the screen.

5. Select all the notebooks that were downloaded in the step 1 of this section.

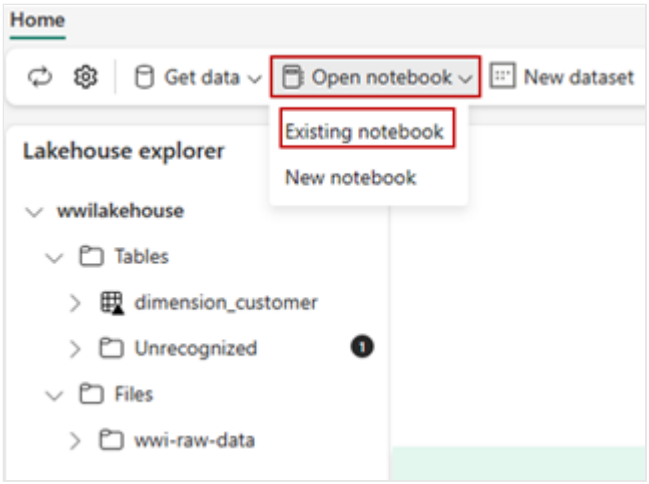
6. Select **Open**. A notification indicating the status of the import appears in the top right corner of the browser window.



7. After the import is successful, you can go to items view of the workspace and see the newly imported notebooks. Select **wwilakehouse** lakehouse to open it.



8. Once the **wwilakehouse** lakehouse is opened, select **Open notebook** > **Existing notebook** from the top navigation menu.



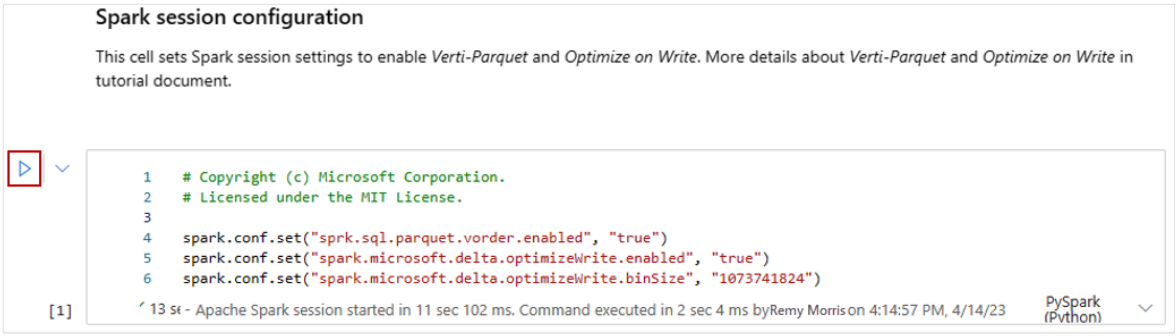
9. From the list of existing notebooks, select the **01 - Create Delta Tables** notebook and select **Open**.

10. In the open notebook in **Lakehouse explorer**, you see the notebook is already linked to your opened lakehouse.

    > **Note**
    >
    > Fabric provides the **V-order** capability to write optimized delta lake files. V-order often improves compression by 3 to 4 times and up to 10 times performance acceleration over the Delta Lake files that aren't optimized. Spark in Fabric dynamically optimizes partitions while generating files with a default 128 MB size. The target file size may be changed per workload requirements using configurations. With the **optimize write** capability, the Apache Spark engine that reduces the number of files written and aims to increase individual file size of the written data.

11. Before you write data as delta lake tables in the **Tables** section of the lakehouse, you use two Fabric features (**V-order** and **Optimize Write**) for optimized data writing and for improved reading performance. To enable these features in your session, set these configurations in the first cell of your notebook.

    To start notebook and all its cells execution in sequence,select **Run All** under **Home** . Or to execute code from that specific cell, you can select the **Run** icon on the left of the cell or press **SHIFT + ENTER** on your keyboard while control is in the cell.



When running a cell, you didn't have to specify the underlying Spark pool or cluster details because Fabric provides them through Live Pool. Every Fabric workspace comes with a default Spark pool, called Live Pool. This means when you create notebooks, you don't have to worry about specifying any Spark configurations or cluster details. When you execute the first notebook command, the live pool is up and running in a few seconds. And the Spark session is established and it starts executing the code. Subsequent code execution is almost instantaneous in this notebook while the Spark session is active.

12. Next, you read raw data from the **Files** section of the lakehouse, and add more columns for different date parts as part of the transformation. Finally, you use partitionBy Spark API to partition the data before writing it as delta table based on the newly created data part columns (Year and Quarter).

```
Python
```

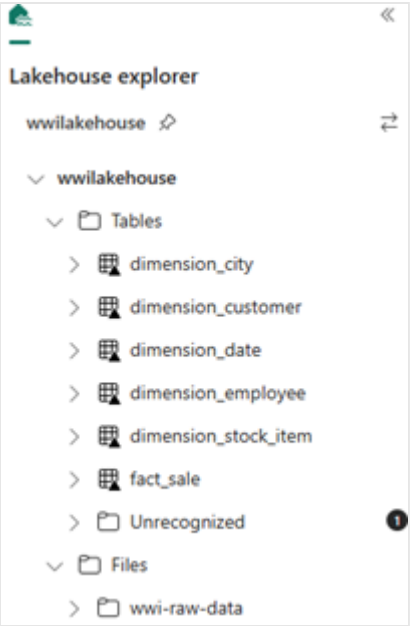
```python
from pyspark.sql.functions import col, year, month, quarter

table_name = 'fact_sale'

df = spark.read.format("parquet").load('Files/wwi-raw-
data/full/fact_sale_1y_full')
df = df.withColumn('Year', year(col("InvoiceDateKey")))
df = df.withColumn('Quarter', quarter(col("InvoiceDateKey")))
df = df.withColumn('Month', month(col("InvoiceDateKey")))

df.write.mode("overwrite").format("delta").partitionBy("Year","Quarter"
).save("Tables/" + table_name)
```

13. After the fact table load, you can move on to loading data for the rest of the dimensions. The following cell creates a function to read raw data from the **Files** section of the lakehouse for each of table names passed as a parameter. Next, it creates a list of dimension tables. Finally, it has a for loop to loop through the list of tables and call created function with each table name as parameter to read data for that specific table and create delta table respectively.

```
Python
```

```
from pyspark.sql.types import *
def loadFullDataFromSource(table_name):
    df = spark.read.format("parquet").load('Files/wwi-raw-data/full/' +
table_name)
    df.write.mode("overwrite").format("delta").save("Tables/" +
table_name)

full_tables = [
    'dimension_city',
    'dimension_date',
    'dimension_employee',
    'dimension_stock_item'
    ]

for table in full_tables:
    loadFullDataFromSource(table)
```

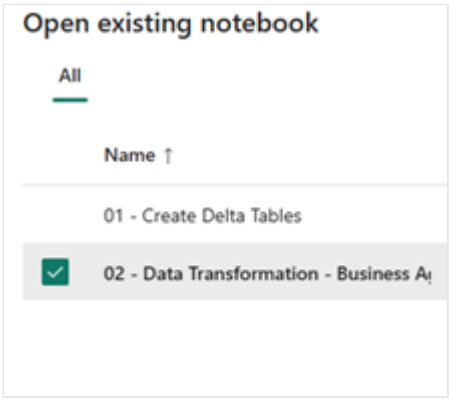
14. To validate the created tables, right click and select refresh on the **wwilakehouse** lakehouse. The tables appear.



15. Go the items view of the workspace again and select the **wwilakehouse** lakehouse to open it.

16. Now, open the second notebook. In the lakehouse view, select **Open notebook** > **Existing notebook** from the ribbon.

17. From the list of existing notebooks, select the **02 - Data Transformation - Business** notebook to open it.



18. In the open notebook in **Lakehouse explorer**, you see the notebook is already linked to your opened lakehouse.

19. An organization might have data engineers working with Scala/Python and other data engineers working with SQL (Spark SQL or T-SQL), all working on the same copy of the data. Fabric makes it possible for these different groups, with varied experience and preference, to work and collaborate. The two different approaches transform and generate business aggregates. You can pick the one suitable for you or mix and match these approaches based on your preference without compromising on the performance:

    - **Approach #1** - Use PySpark to join and aggregates data for generating business aggregates. This approach is preferable to someone with a programming (Python or PySpark) background.

    - **Approach #2** - Use Spark SQL to join and aggregates data for generating business aggregates. This approach is preferable to someone with SQL background, transitioning to Spark.

20. **Approach #1 (sale_by_date_city)** - Use PySpark to join and aggregate data for generating business aggregates. With the following code, you create three different Spark dataframes, each referencing an existing delta table. Then you join these tables using the dataframes, do group by to generate aggregation, rename a few of the columns, and finally write it as a delta table in the **Tables** section of the lakehouse to persist with the data.

    In this cell, you create three different Spark dataframes, each referencing an existing delta table.

    Python

    ```
    df_fact_sale = spark.read.table("wwilakehouse.fact_sale")
    df_dimension_date = spark.read.table("wwilakehouse.dimension_date")
    df_dimension_city = spark.read.table("wwilakehouse.dimension_city")
    ```

In this cell, you join these tables using the dataframes created earlier, do group by to generate aggregation, rename a few of the columns, and finally write it as a delta table in the **Tables** section of the lakehouse.

Python

```
sale_by_date_city = df_fact_sale.alias("sale") \
.join(df_dimension_date.alias("date"), df_fact_sale.InvoiceDateKey ==
df_dimension_date.Date, "inner") \
.join(df_dimension_city.alias("city"), df_fact_sale.CityKey ==
df_dimension_city.CityKey, "inner") \
.select("date.Date", "date.CalendarMonthLabel", "date.Day",
"date.ShortMonth", "date.CalendarYear", "city.City",
"city.StateProvince", "city.SalesTerritory", "sale.TotalExcludingTax",
"sale.TaxAmount", "sale.TotalIncludingTax", "sale.Profit")\
.groupBy("date.Date", "date.CalendarMonthLabel", "date.Day",
"date.ShortMonth", "date.CalendarYear", "city.City",
"city.StateProvince", "city.SalesTerritory")\
.sum("sale.TotalExcludingTax", "sale.TaxAmount",
"sale.TotalIncludingTax", "sale.Profit")\
.withColumnRenamed("sum(TotalExcludingTax)", "SumOfTotalExcludingTax")\
.withColumnRenamed("sum(TaxAmount)", "SumOfTaxAmount")\
.withColumnRenamed("sum(TotalIncludingTax)", "SumOfTotalIncludingTax")\
.withColumnRenamed("sum(Profit)", "SumOfProfit")\
.orderBy("date.Date", "city.StateProvince", "city.City")

sale_by_date_city.write.mode("overwrite").format("delta").option("overw
riteSchema", "true").save("Tables/aggregate_sale_by_date_city")
```

21. **Approach #2 (sale_by_date_employee)** - Use Spark SQL to join and aggregate data for generating business aggregates. With the following code, you create a temporary Spark view by joining three tables, do group by to generate aggregation, and rename a few of the columns. Finally, you read from the temporary Spark view and finally write it as a delta table in the **Tables** section of the lakehouse to persist with the data.

    In this cell, you create a temporary Spark view by joining three tables, do group by to generate aggregation, and rename a few of the columns.

    Python

    ```
    %%sql
    CREATE OR REPLACE TEMPORARY VIEW sale_by_date_employee
    AS
    SELECT
           DD.Date, DD.CalendarMonthLabel
     , DD.Day, DD.ShortMonth Month, CalendarYear Year
          ,DE.PreferredName, DE.Employee
          ,SUM(FS.TotalExcludingTax) SumOfTotalExcludingTax
    ```

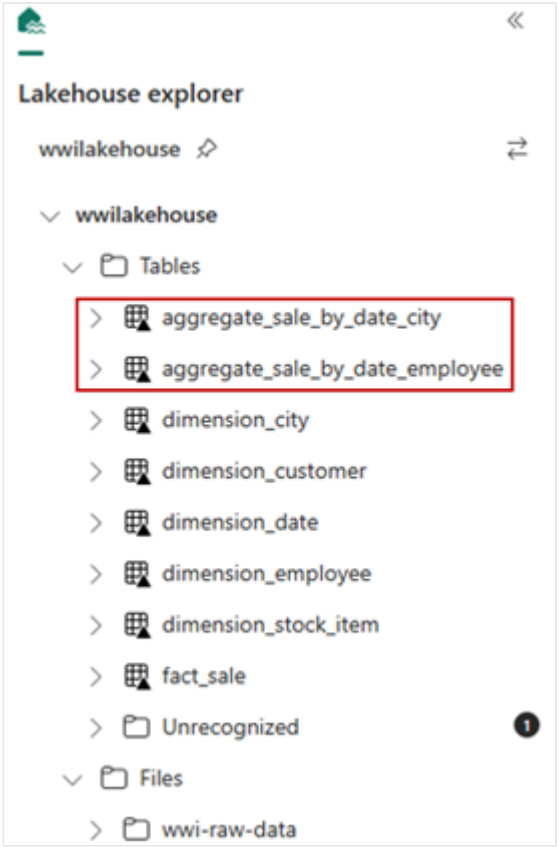
```
      ,SUM(FS.TaxAmount) SumOfTaxAmount
      ,SUM(FS.TotalIncludingTax) SumOfTotalIncludingTax
      ,SUM(Profit) SumOfProfit
FROM wwilakehouse.fact_sale FS
INNER JOIN wwilakehouse.dimension_date DD ON FS.InvoiceDateKey =
DD.Date
INNER JOIN wwilakehouse.dimension_Employee DE ON FS.SalespersonKey =
DE.EmployeeKey
GROUP BY DD.Date, DD.CalendarMonthLabel, DD.Day, DD.ShortMonth,
DD.CalendarYear, DE.PreferredName, DE.Employee
ORDER BY DD.Date ASC, DE.PreferredName ASC, DE.Employee ASC
```

In this cell, you read from the temporary Spark view created in the previous cell and finally write it as a delta table in the **Tables** section of the lakehouse.

Python

```
sale_by_date_employee = spark.sql("SELECT * FROM
sale_by_date_employee")
sale_by_date_employee.write.mode("overwrite").format("delta").option("o
verwriteSchema", "true").save("Tables/aggregate_sale_by_date_employee")
```

22. To validate the created tables, right click and select refresh on the **wwilakehouse** lakehouse. The aggregate tables appear.



Both the approaches produce a similar outcome. You can choose based on your background and preference, to minimize the need for you to learn a new technology or

compromise on the performance.

Also you may notice that you're writing data as delta lake files. The automatic table discovery and registration feature of Fabric picks up and registers them in the metastore. You don't need to explicitly call `CREATE TABLE` statements to create tables to use with SQL.

Next steps

Advance to the next article to learn about

Build reports using Power BI

Lakehouse tutorial: Building reports in Microsoft Fabric

Article • 05/23/2023

In this section of the tutorial, you create a Power BI data model and create a report from scratch.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Prerequisites

- Prepare and transform the data using notebooks and Spark runtime

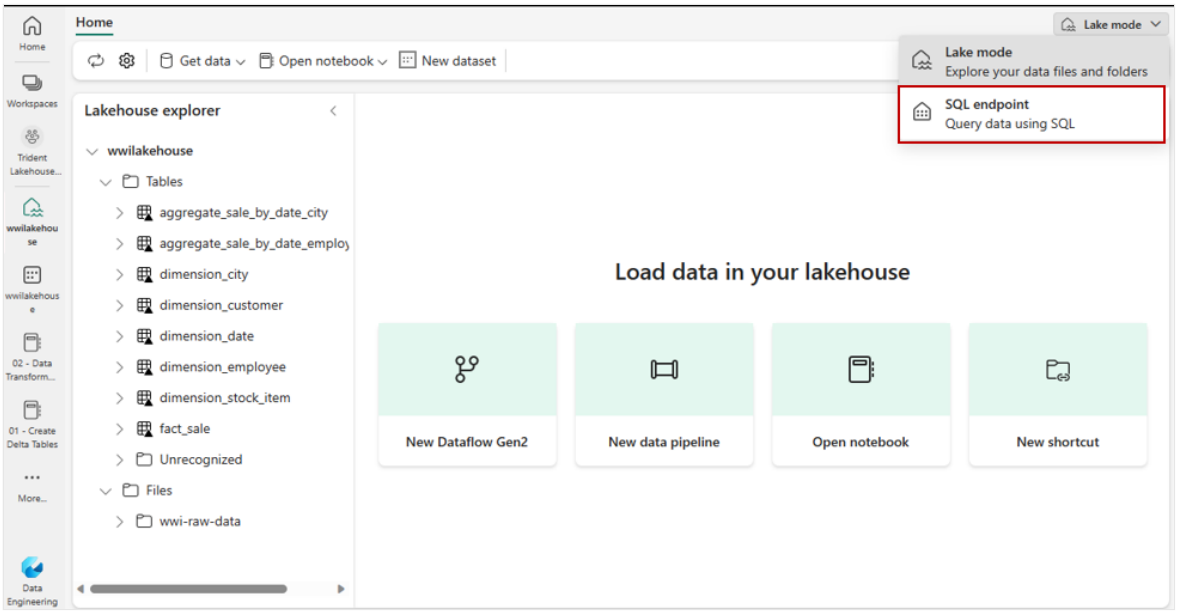
Build a report

Power BI is natively integrated in the whole Fabric experience. This native integration brings a unique mode, called DirectLake, of accessing the data from the lakehouse to provide the most performant query and reporting experience. DirectLake mode is a groundbreaking new engine capability to analyze very large datasets in Power BI. The technology is based on the idea of loading parquet-formatted files directly from a data lake without having to query a data warehouse or lakehouse endpoint, and without having to import or duplicate data into a Power BI dataset. DirectLake is a fast path to load the data from the data lake straight into the Power BI engine, ready for analysis.

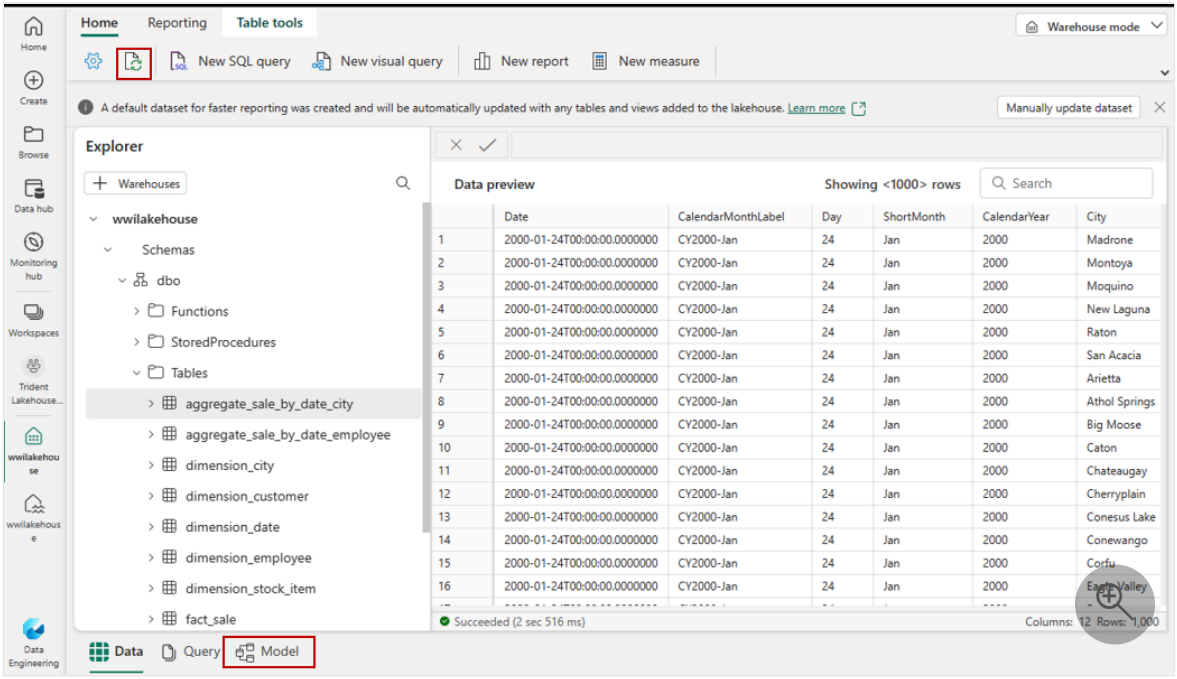
In traditional DirectQuery mode, the Power BI engine queries the data directly from the data source every time it's queried and hence query performance depends on the speed data can be retrieved from the data source. This method avoids having to copy the data; any changes at the source are immediately reflected in the query results while in the import mode. And yet performance is better because the data is readily available in memory without having to query the data source each time. However, the Power BI engine must first copy the data into the dataset at refresh time. Any changes at the source are only picked up during the next data refresh.

DirectLake mode now eliminates this import requirement by loading the data files directly into memory. Because there's no explicit import process, it's possible to pick up any changes at the source as they occur, thus combining the advantages of DirectQuery and import mode while avoiding their disadvantages. DirectLake mode is therefore the ideal choice for analyzing very large datasets and datasets with frequent updates at the source.

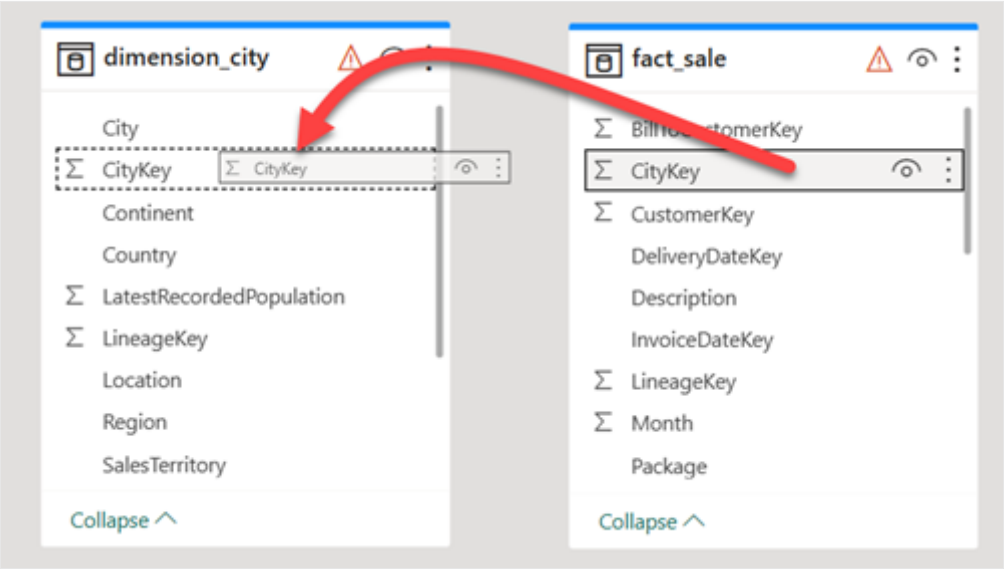
1. Go the **wwilakehouse** lakehouse, select **SQL endpoint** under **Lake mode** on top right of the screen to open warehouse mode of the selected lakehouse.



2. Once you are in warehouse mode, you should be able to see all the tables you created. If you don't see them yet, select the **Refresh** icon at the top. Next, select the **Model** tab at the bottom to open the default Power BI dataset.



3. For this data model, you need to define the relationship between different tables so that you can create reports and visualizations based on data coming across different tables. From the **fact_sale** table, drag the **CityKey** field and drop it on the **CityKey** field in the **dimension_city** table to create a relationship. Check the "Assume referential integrity" and select ***Confirm*** to establish the relationship.
Note - When defining relationships, make sure you have many to one relationship from the fact to the dimension and not vice versa.



4. On the **Create Relationship** settings:

   a. Table 1 is populated with fact_sale and the column of CityKey.

   b. Table 2 is populated with dimension_city and the column of CityKey.

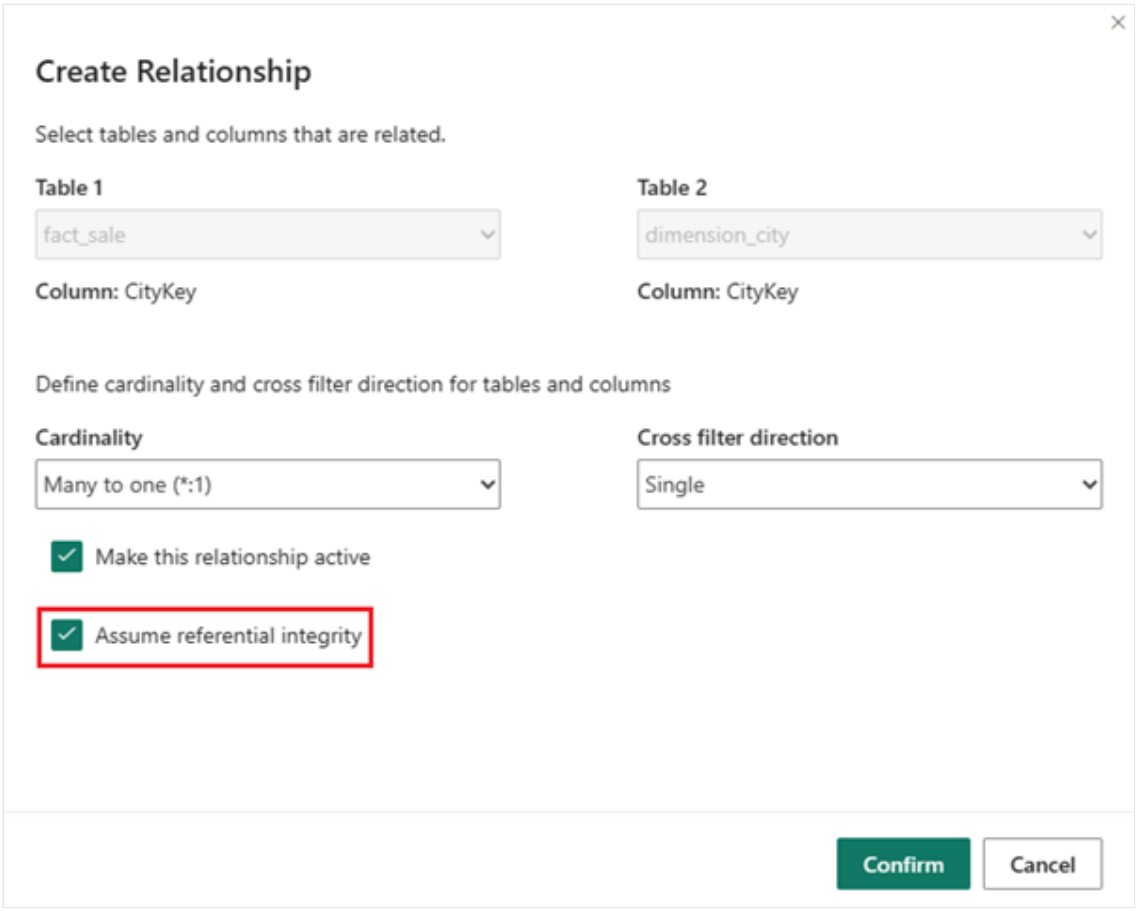
   c. Cardinality: **Many to one (*:1)**

   d. Cross filter direction: **Single**

   e. Leave the box next to **Make this relationship active** checked.

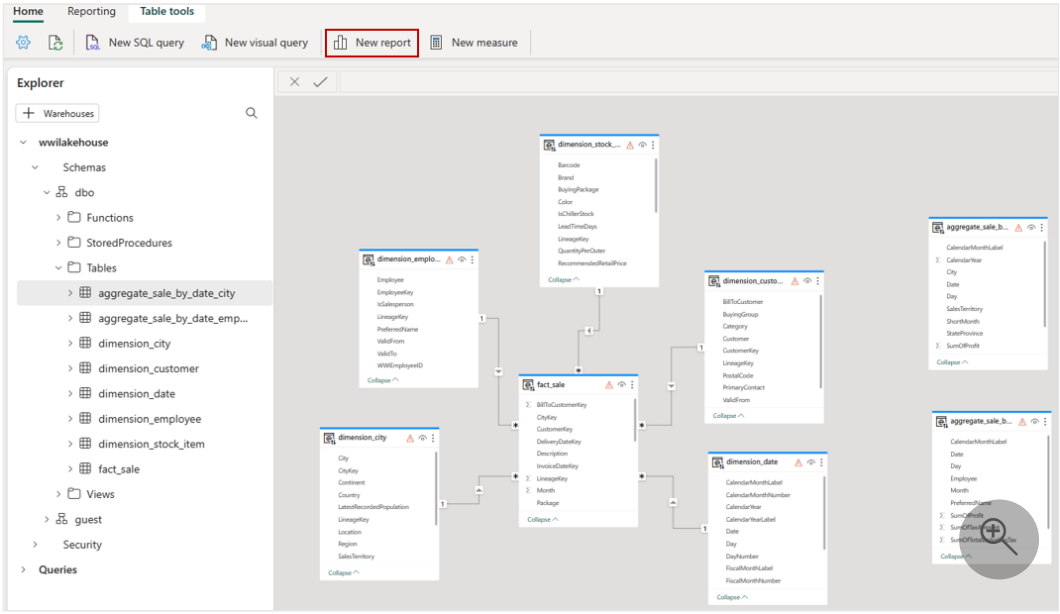
   f. Check the box next to **Assume referential integrity.**

   g. Select **Confirm.**

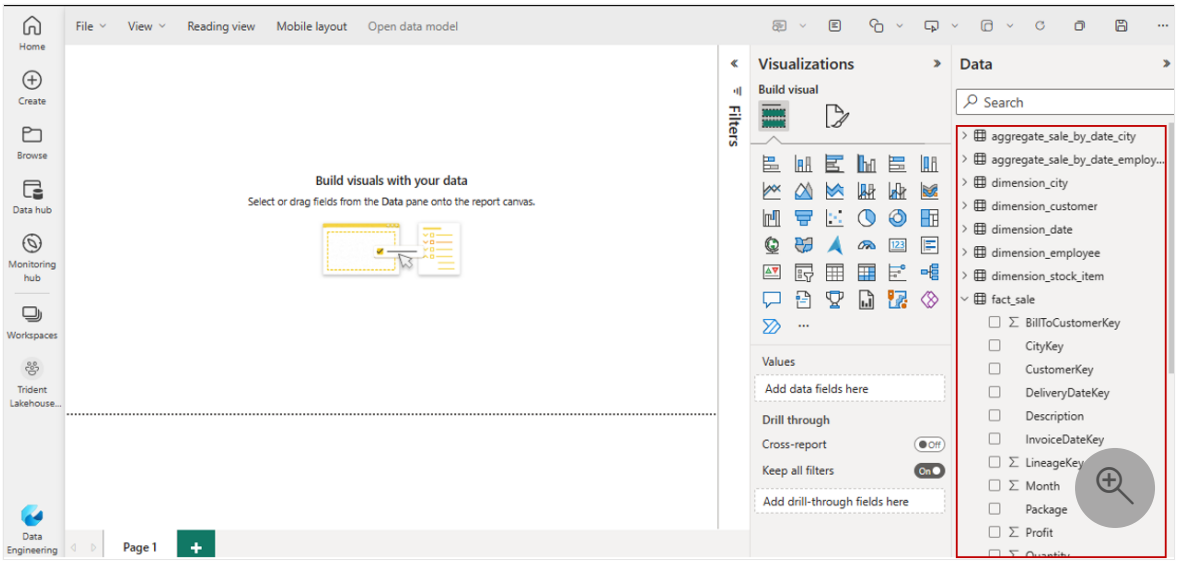


h. Similarly, you need to add these relationships as well. After you add these relationships, your data model is ready for reporting as shown in the previous image:

- StockItemKey(fact_sale) - StockItemKey(dimension_stock_item)
- Salespersonkey(fact_sale) - EmployeeKey(dimension_employee)
- CustomerKey(fact_sale) - CustomerKey(dimension_customer)
- InvoiceDateKey(fact_sale) - Date(dimension_date)



5. Select **New report** to start creating reports/dashboards in Power BI. On the Power BI report canvas, you can create reports to meet your business requirements by dragging required columns from the **Data** pane to the canvas and using one or more of available visualizations.



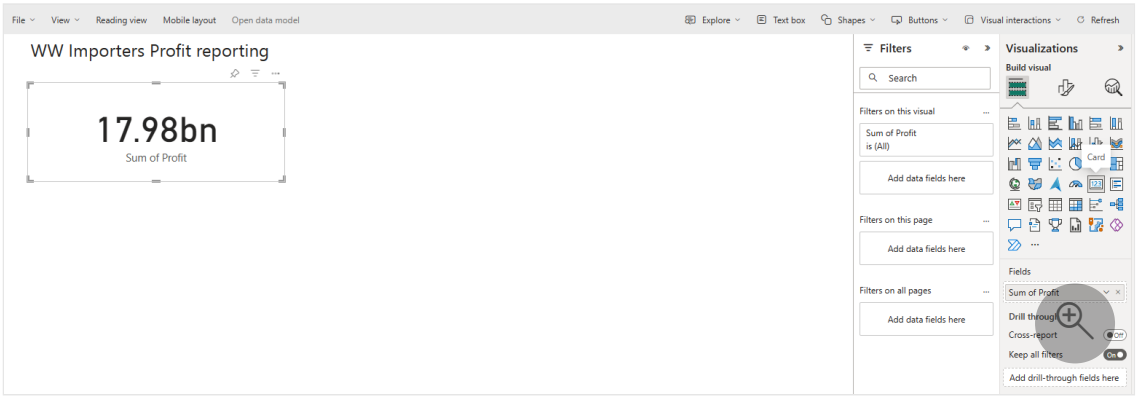
6. Add a title:
   a. In the Ribbon, select the Text Box icon.
   b. Type in **WW Importers Profit Reporting**.
   c. Highlight the text and increase size to 20 and place in the upper left of the report page.

7. Add a Card:

   a. On the **Data** pane, expand **fact_sales** and check the box next to **Profit**. This selection creates a column chart and adds the field to the Y-axis.

   b. With the bar chart selected, select the Card visual in the visualization pane. This selection converts the visual to a card.

   c. Place the card under the title.

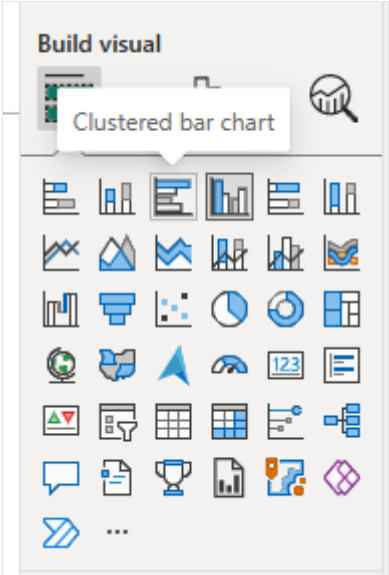


8. Add a Bar chart:

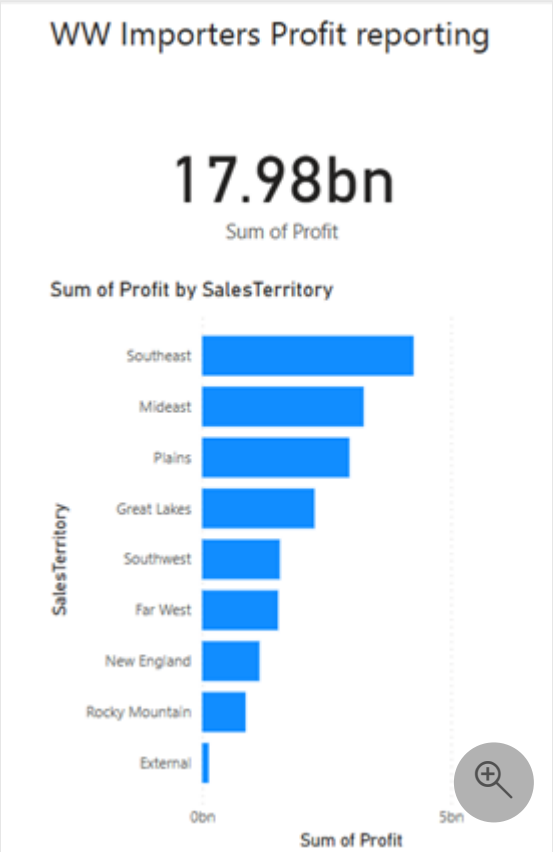
a. On the **Data** pane, expand **fact_sales** and check the box next to **Profit**. This selection creates a column chart and adds the field to the Y-axis.

b. On the **Data** pane, expand **dimension_city** and check the box for **SalesTerritory**. This selection adds the field to the Y-axis.

c. With the bar chart selected, select the Clustered Bar Chart visual in the visualization pane. This selection converts the column chart into a bar chart.



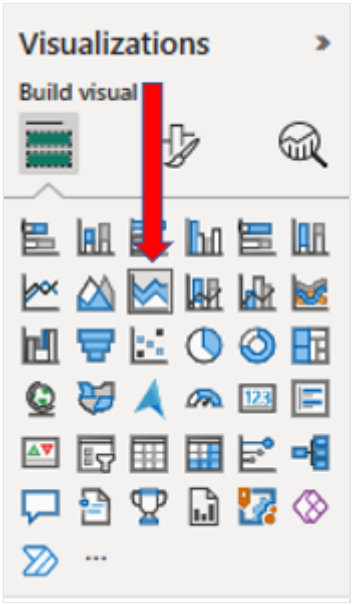
d. Resize the Bar chart to fill in the area under the title and Card.



9. Click anywhere on the blank canvas (or press the Esc key) so the bar chart visual is no longer selected.

10. Build a stacked area chart visual:

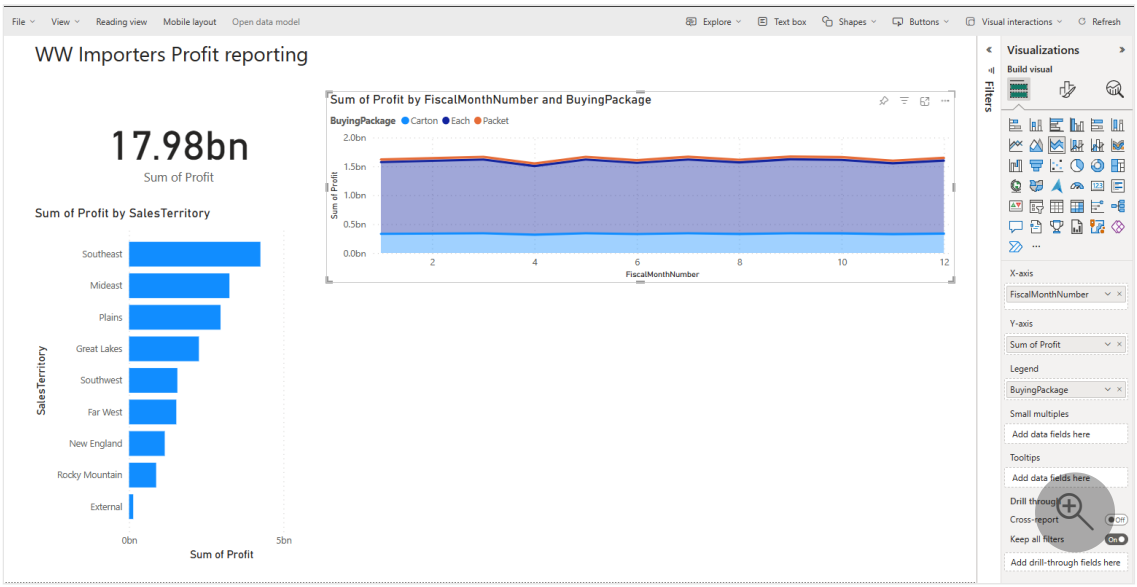
    a. On the **Visualizations** pane, select the **Stacked area chart** visual.



    b. Reposition and resize the stacked area chart to the right of the card and bar chart visuals created in the previous steps.

    c. On the **Data** pane, expand **fact_sales** and check the box next to **Profit.** Expand dimension_date and check the box next to **FiscalMonthNumber**. This selection creates a filled line chart showing profit by fiscal month.

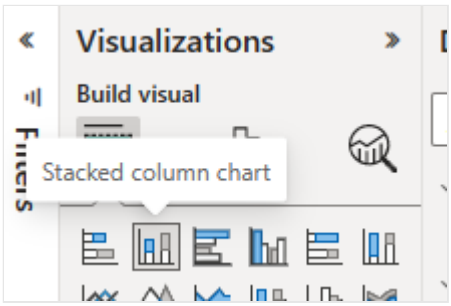
    d. On the **Data** pane, expand **dimension_stock_item** and drag **BuyingPackage** into the Legend field well. This selection adds a line for each of the Buying Packages.



11. Click anywhere on the blank canvas (or press the Esc key) so the stacked area chart visual is no longer selected.

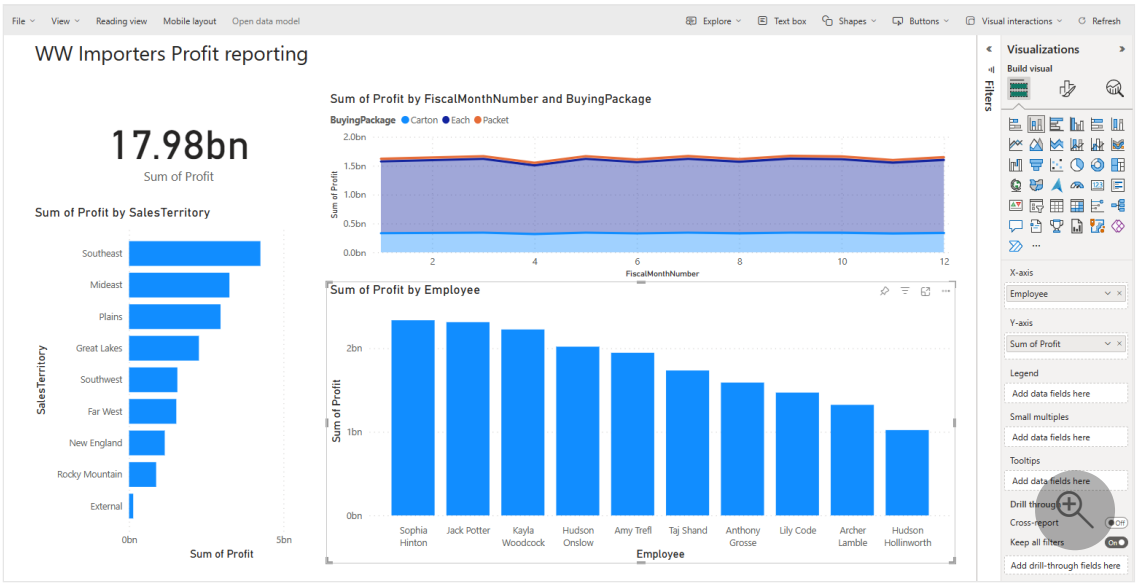
12. Build a column chart:

    a. On the **Visualizations** pane, select the **Stacked column chart** visual.



    b. On the **Data** pane, expand **fact_sales** and check the box next to **Profit**. This selection adds the field to the Y-axis.

    c. On the **Data** pane, expand **dimension_employee** and check the box next to **Employee**. This selection adds the field to the X-axis.

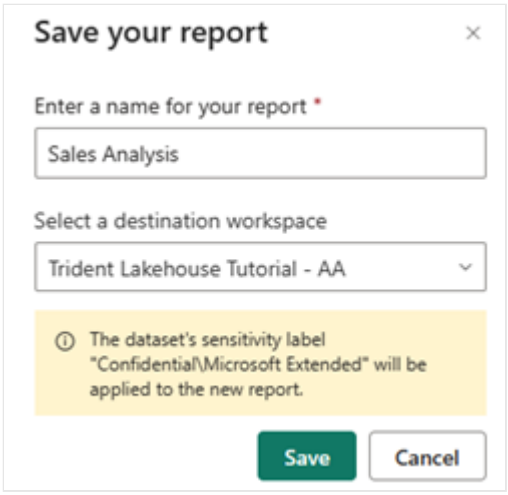


13. Click anywhere on the blank canvas (or press the Esc key) so the LINE chart visual is no longer selected.

14. From the ribbon, select **File** > **Save**.

15. Enter the name of your report as **Profit Reporting**.

16. Select **Save**.



Next steps

Advance to the next article to learn how to

Clean up resources

Lakehouse tutorial: Clean up Fabric resources

Article • 05/23/2023

As a final step in the tutorial, clean up your resources. This article shows how to delete the workspace.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

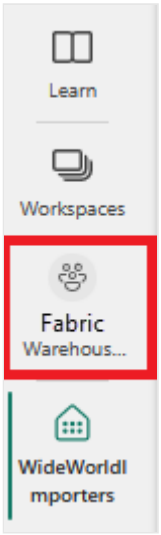
Prerequisites

- Create the Power BI data model and a report

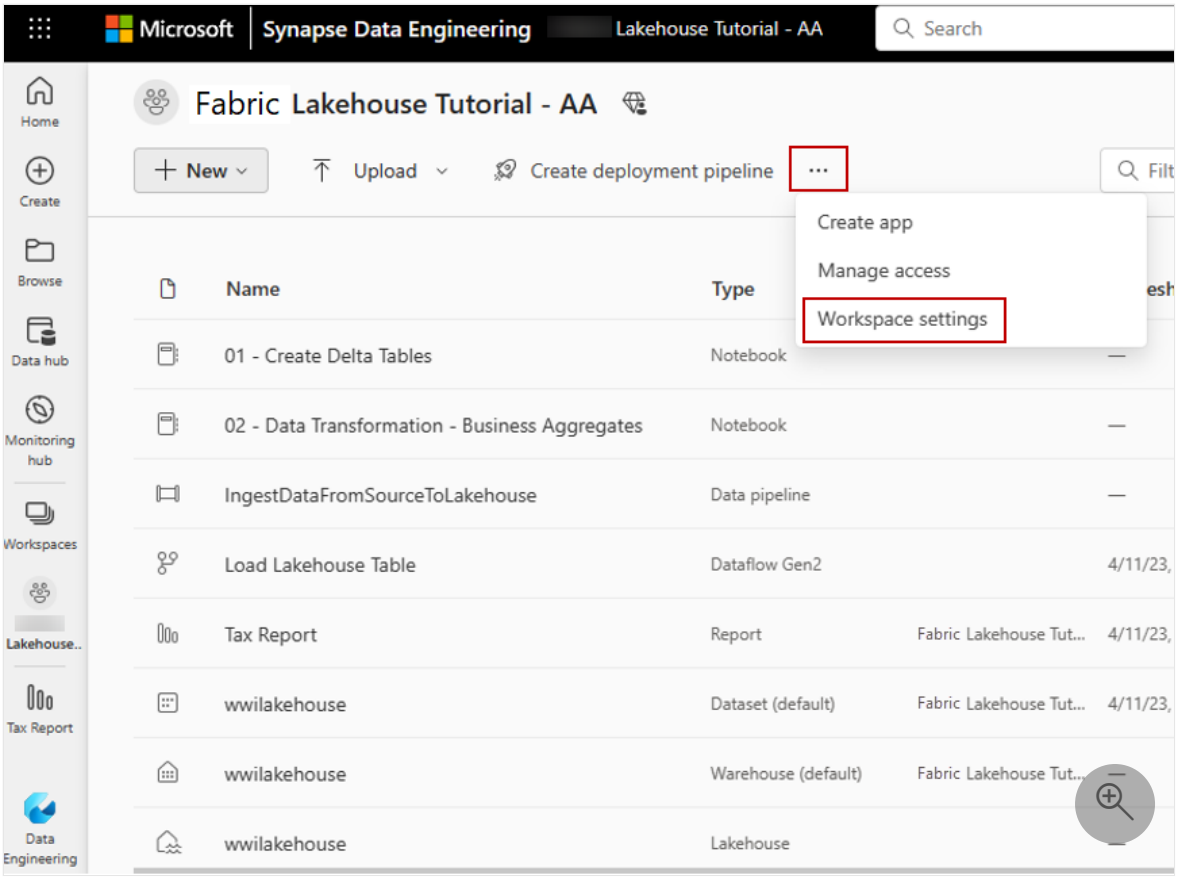
Clean up resources

You can delete individual reports, pipelines, warehouses, and other items or remove the entire workspace. Use the following steps to delete the workspace you created for this tutorial:

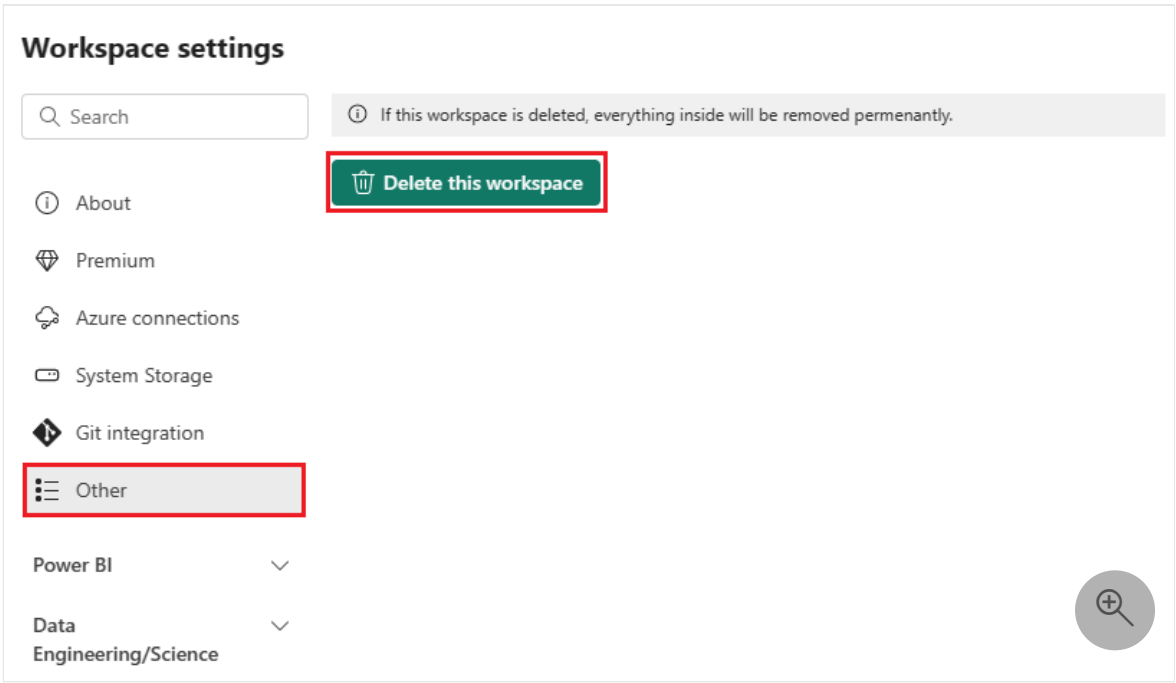
1. Select your workspace, the **Fabric Lakehouse Tutorial** from the left-hand navigation menu. It opens the workspace item view.



2. Select the ... option under the workspace name and select **Workspace settings**.



3. Select **Other** and **Delete this workspace**.



4. When the warning pops up, select **Delete**.

Next steps

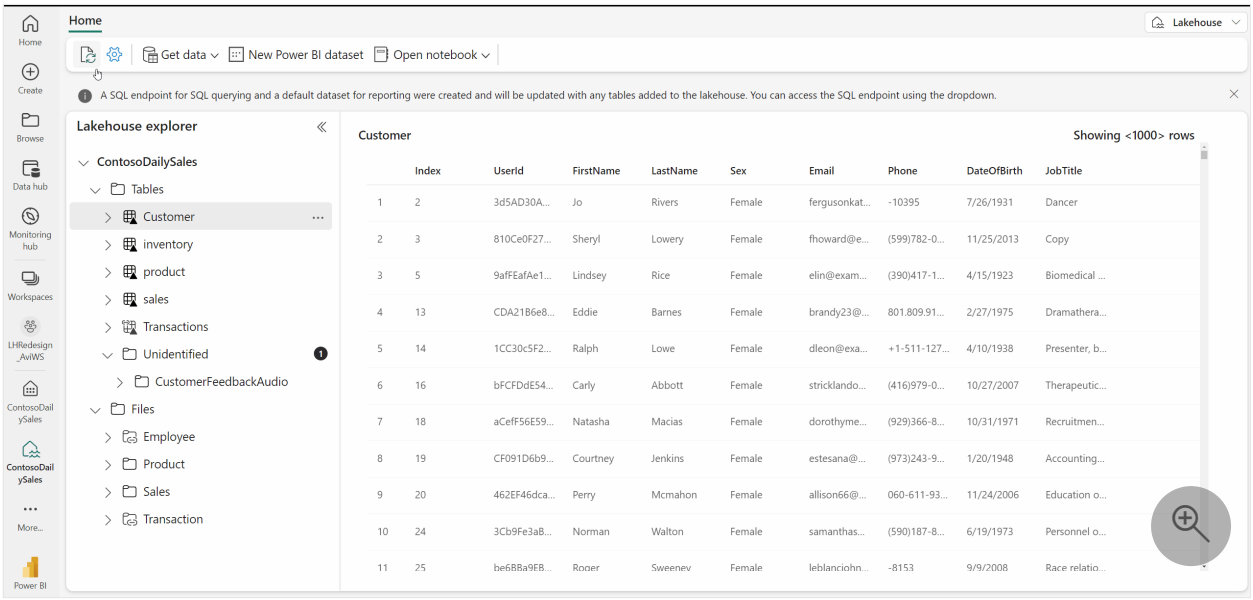
Advance to the next article to learn about

Options to get data into lakehouse

What is a lakehouse in Microsoft Fabric?

Article • 05/23/2023

Microsoft Fabric Lakehouse is a data architecture platform for storing, managing, and analyzing structured and unstructured data in a single location. It is a flexible and scalable solution that allows organizations to handle large volumes of data using a variety of tools and frameworks to process and analyze that data. It integrates with other data management and analytics tools to provide a comprehensive solution for data engineering and analytics.



> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Lakehouse SQL endpoint

The Lakehouse creates a serving layer by auto-generating an SQL endpoint and a default dataset during creation. This new see-through functionality allows user to work directly on top of the delta tables in the lake to provide a frictionless and performant experience all the way from data ingestion to reporting.

An important distinction between default warehouse is that it's a read-only experience and doesn't support the full T-SQL surface area of a transactional data warehouse. It is important to note that only the tables in Delta format are available in the SQL Endpoint.

Parquet, CSV, and other formats can't be queried using the SQL Endpoint. If you don't see your table, convert it to Delta format.

Learn more about the SQL Endpoint here

Automatic table discovery and registration

The automatic table discovery and registration is a feature of Lakehouse that provides a fully managed file to table experience for data engineers and data scientists. You can drop a file into the managed area of the Lakehouse and the file is automatically validated for supported structured formats, which currently is only Delta tables, and registered into the metastore with the necessary metadata such as column names, formats, compression and more. You can then reference the file as a table and use SparkSQL syntax to interact with the data.

Interacting with the Lakehouse item

A data engineer can interact with the lakehouse and the data within the lakehouse in several ways:

- **The Lakehouse explorer**: The explorer is the main Lakehouse interaction page. You can load data in your Lakehouse, explore data in the Lakehouse using the object explorer, set MIP labels & various other things. Learn more about the explorer experience: Navigating the Lakehouse explorer.

- **Notebooks**: Data engineers can use the notebook to write code to read, transform and write directly to Lakehouse as tables and/or folders. You can learn more about how to leverage notebooks for Lakehouse: Explore the data in your Lakehouse with a notebook and How to use a notebook to load data into your Lakehouse.

- **Pipelines**: Data engineers can use data integration tools such as pipeline copy tool to pull data from other sources and land into the Lakehouse. Find more information on how to use the copy activity: How to copy data using copy activity.

- **Apache Spark job definitions**: Data engineers can develop robust applications and orchestrate the execution of compiled Spark jobs in Java, Scala, and Python. Learn more about Spark jobs: What is an Apache Spark job definition?.

- **Dataflows Gen 2**: Data engineers can leverage Dataflows Gen 2 to ingest and prepare their data. Find more information on load data using dataflows: Create your first dataflow to get and transform data.

Learn more about the different ways to load data into your lakehouse: Get data experience for Lakehouse.

Next steps

In this overview, you get a basic understanding of a lakehouse. Advance to the next article to learn how to create and get started with your own lakehouse:

- To get started with lakehouse, see Creating a lakehouse.

Create a lakehouse in Microsoft Fabric

Article • 05/23/2023

In this tutorial, you learn different ways to create a Lakehouse in Microsoft Fabric.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Create a lakehouse

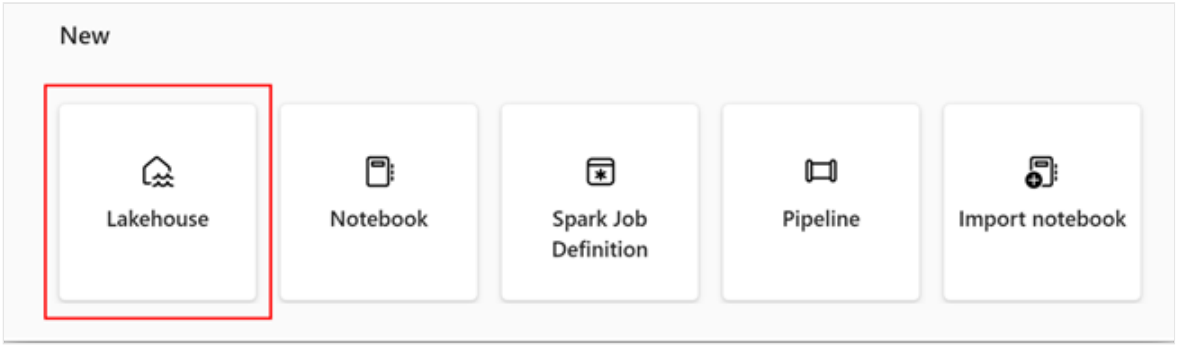
The lakehouse creation process is quick and simple; there are several ways to get started.

Ways to create a lakehouse

There are a few ways you can get started with the creation process:

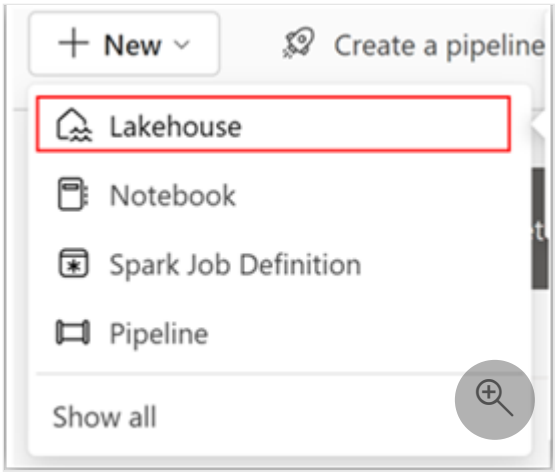
1. **Data Engineering** homepage

    - You can easily create a lakehouse through the **Lakehouse** card under the **New** section in the homepage.



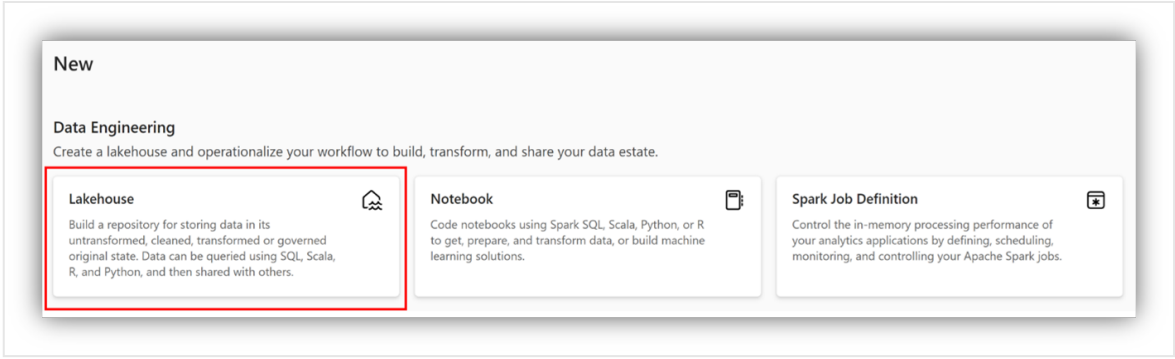
2. **Workspace** view

    - You can also create a lakehouse through the workspace view when you are on the **Data Engineering** experience by using the **New** dropdown.



3. **Create Hub**

    - An entry point to create a lakehouse is available in the **Create Hub** page under **Data Engineering**.



Creating a lakehouse from the Data Engineering homepage

1. Browse to the **Data Engineering** homepage.
2. Under the New section, locate the **Lakehouse** card and select it to get started with the creation process
3. Enter a name for the lakehouse and a sensitivity label if your organization requires one, and select **Create**.
4. Once the lakehouse is created, you land on the **Lakehouse Editor** page where you can get started and load data.

> ⓘ **Note**
>
> The lakehouse will be created under the current workspace you are in.

Next steps

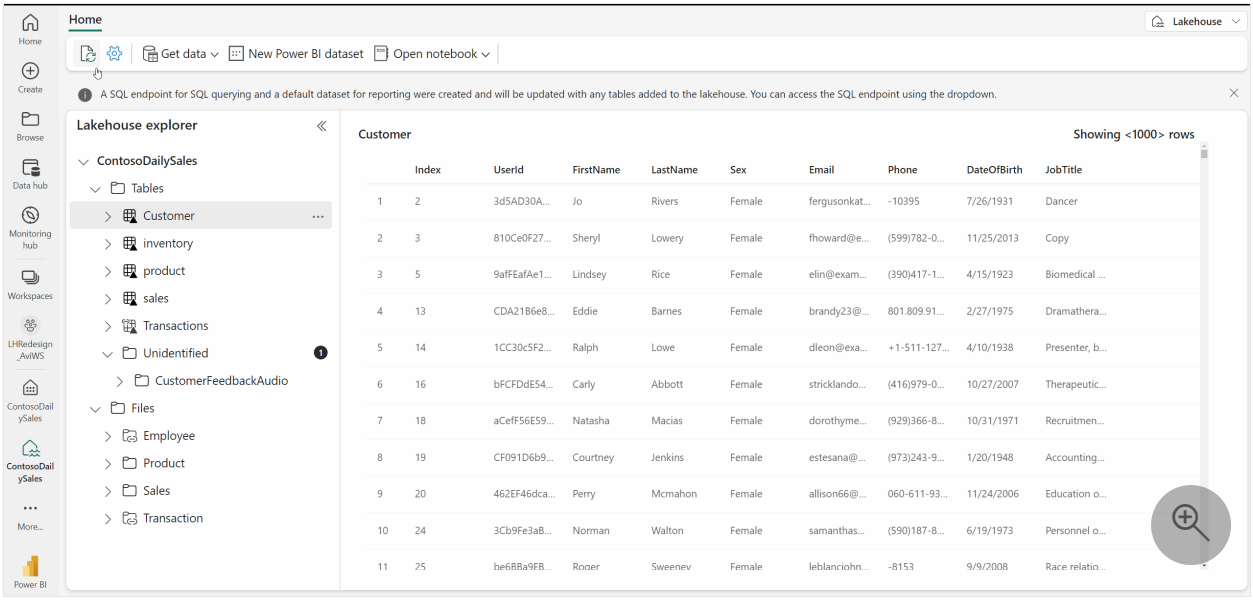
Now that you have successfully created your Lakehouse, learn more about:

- Different ways to load data in Lakehouse, see Get data experience for Lakehouse
- Exploring your lakehouse explorer, see Navigating the Lakehouse explorer

Navigate the Fabric Lakehouse explorer

Article • 05/23/2023

The Lakehouse explorer page is the main Lakehouse interaction page; you can use it to load data into your Lakehouse, browse through the data, preview them, and many other things. The page is divided into three sections: the Lakehouse explorer, the main view, and the ribbon.



> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Lakehouse explorer

The Lakehouse explorer provides a unified graphical representation of the whole Lakehouse for users to navigate, access, and update their data.

- The **Table Section** is a UI representation of the managed area of your lake which is typically organized and governed to facilitate efficient data processing and analysis. All tables, whether automatically or explicitly created and registered in SyMS, are displayed here. You can preview table data, view the table schema, access underlying files of selected table, and perform various other actions.
- The **Unidentified Area** is a part of the managed area of your lake which displays any folders or files in the managed area with no associated tables in SyMS. For

example, if a user were to drop an unsupported folder/file in the managed area, eg: a directory full of pictures or audio, this would not get picked up by our auto-detection process and hence not have an associated table. In this case, this folder would be found in this unidentified area. The main purpose of this new section is to promote either deleting these files from the managed area or moving them to the file section for further processing.

- The **File Section** is the UI representation of the unidentified area of your lake, it can be seen as a "landing zone" for raw data that is ingested from various sources and requires additional processing before it can be used for analysis. You can navigate through directories, preview files, load a file into a table and perform various other actions.

Main view area

The main view area of the Lakehouse page is the space where most of the data interaction occurs. The view changes depending on what you select. Since the object explorer only displays a folder level hierarchy of the lake, the main view area is what you use to navigate your files, preview files, and various other tasks.

Ribbon

The Lakehouse ribbon is a quick go-to action bar for you to refresh the Lakehouse, update settings, load data or create a new dataset.

Different ways to load data into a Lakehouse

There are several ways to load data into your Lakehouse from the explorer page:

- **Local file/folder upload:** Uploading data directly from your local machine to the File section of your Lakehouse.
- **Notebook code:** Using available spark libraries to connect to a data source directly and then loading data to dataframe and saving it in your Lakehouse.
- **Copy tool in pipelines:** Connect to different data sources and land the data either in original format or convert it to a delta table.
- **Dataflows Gen 2:** Creating a dataflow to get data in, then transform and publish it into your Lakehouse.

- **Shortcuts:** Creating shortcuts to connect to existing data into your Lakehouse without having to directly copy it.

Next steps

- Learn more about the different use cases to understand the best way to load your data, see Get data experience for Lakehouse.
- Explore the data in your Lakehouse with a notebook
- How to use a notebook to load data into your Lakehouse.
- To get started with Pipelines copy activity, see How to copy data using copy activity.
- Create your first dataflow to get and transform data.
- Create a OneLake shortcut.

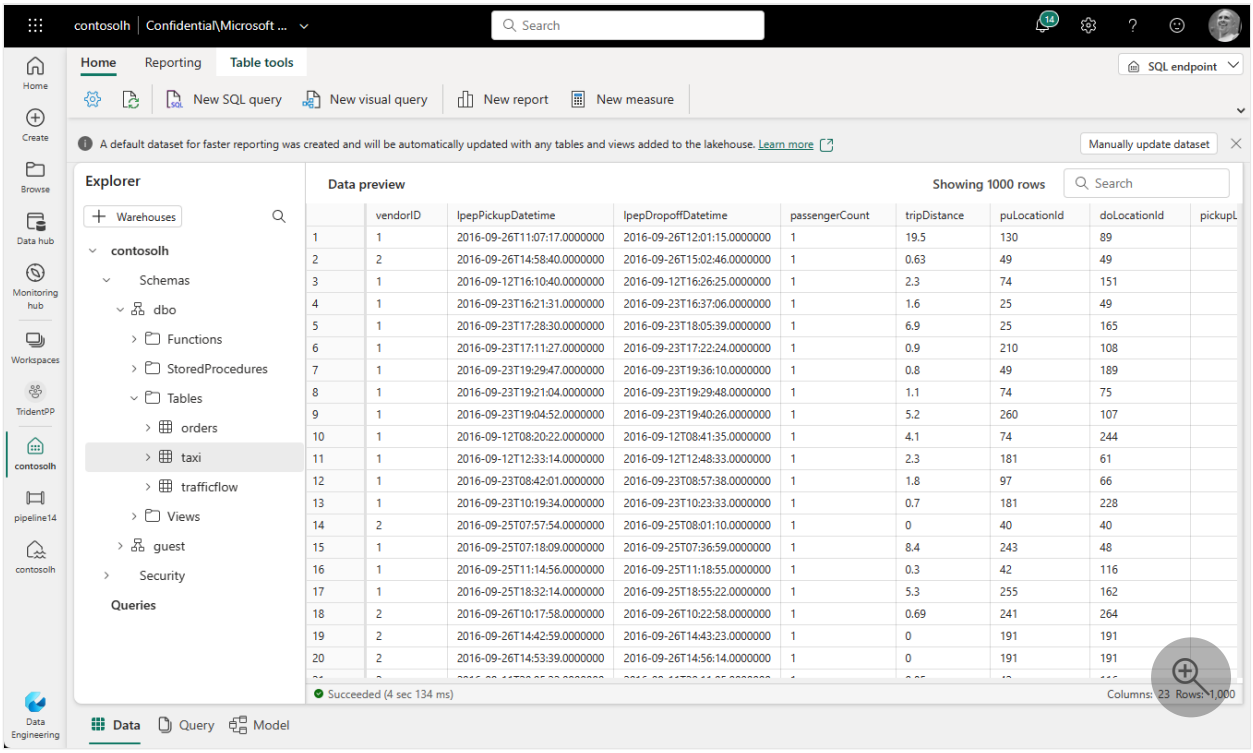
What is SQL Endpoint for a lakehouse?

Article • 05/23/2023

Microsoft Fabrics provides a SQL-based experience for lakehouse delta tables. This SQL-based experience is called the SQL Endpoint. You can analyze data in delta tables using T-SQL language, save functions, generate views, and apply SQL security. To access SQL Endpoint, you select a corresponding item in the workspace view or switch to SQL endpoint mode in Lakehouse Explorer.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Creating a lakehouse creates a SQL endpoint, which points to the lakehouse delta table storage. Once you create a delta table in the Lakehouse, it's immediately available for querying using the SQL endpoint. To learn more, see Data Warehouse documentation: SQL Endpoint



SQL endpoint read-only mode

SQL endpoint operates in read-only mode over lakehouse delta tables. You can only read data from delta tables using SQL endpoint. They can save functions, views, and set

SQL object-level security.

> **Note**
>
> External delta tables created with Spark code won't be visible to SQL endpoint. Use shortcuts in Table space to make external delta tables visible to SQL endpoint.

To modify data in lakehouse delta tables, you have to switch to lakehouse mode and use Apache Spark.

Access control using SQL security

You can set object-level security for accessing data using SQL endpoint. These security rules will only apply for accessing data via SQL Endpoint. To ensure data is not accessible in other ways, you must set workspace roles and permissions, see Workspace roles and permissions.

Next steps

- Get started with the SQL Endpoint of the Lakehouse in Microsoft Fabric
- Workspace roles and permissions
- Security for data warehousing in Microsoft Fabric

How direct lake mode works with PBI reporting

Article • 05/23/2023

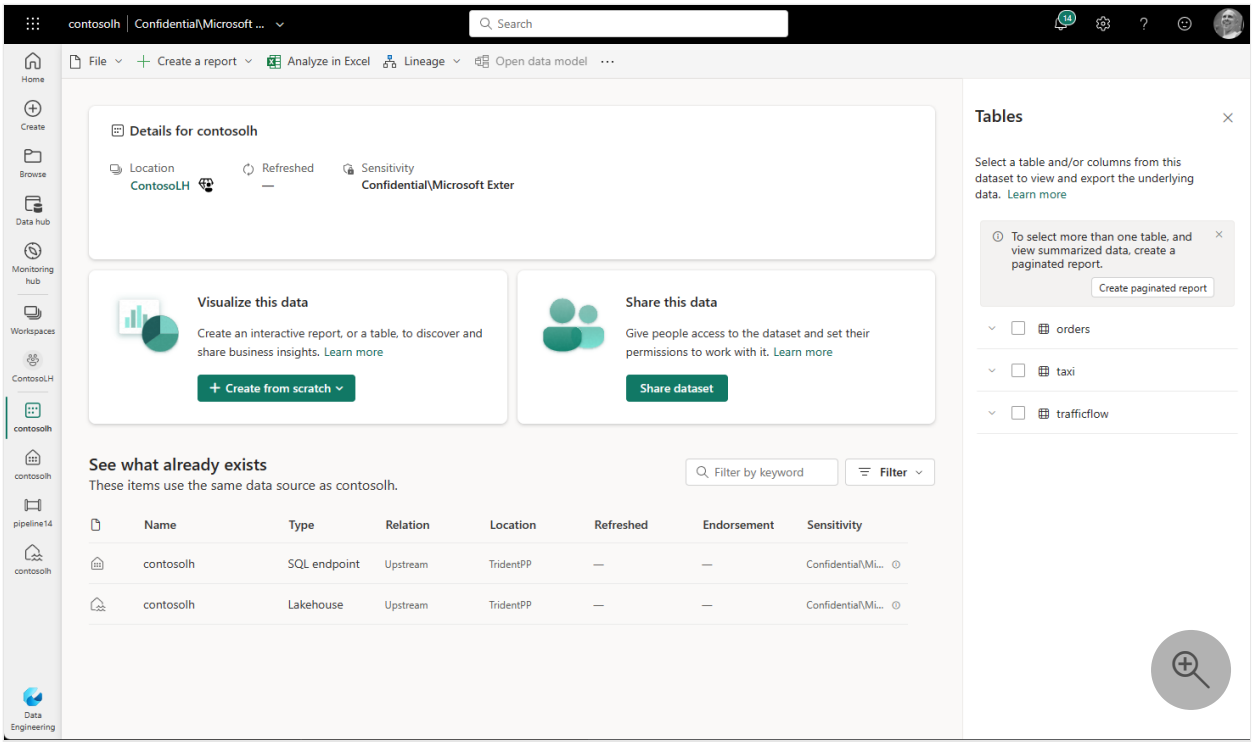
In Microsoft Fabric, when the user creates a lakehouse, the system also provisions the associated SQL endpoint and default dataset. The default dataset is a semantic model with metrics on top of lakehouse data. The dataset allows Power BI to load data for reporting.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

When a Power BI report shows an element that uses data, it requests it from the underlying dataset. Next the dataset accesses a lakehouse to retrieve data and return it to the Power BI report. For efficiency default dataset loads commonly requested data into the cache and refreshes it when needed.

Lakehouse applies V-order optimization to tables. This optimization enables quickly loading data into the dataset and having it ready for querying without any other sorting or transformations.

This approach gives unprecedented performance and the ability to instantly load large amounts of data for Power BI reporting.



Setting permissions for report consumption

The default dataset is retrieving data from a lakehouse on demand. To make sure that data is accessible for the user that is viewing Power BI report, necessary permissions on the underlying lakehouse need to be set.

One option is to give the user *Viewer* role in the workspace and grant necessary permissions to data using SQL endpoint security. Alternatively, the user can be given *Admin, Member, or Contributor* role to have full access to the data.

Next steps

- Default Power BI datasets in Microsoft Fabric

Options to get data into the Fabric Lakehouse

Article • 05/23/2023

Get data experience covers all user scenarios for bringing data into the lakehouse, like:

- Connecting to existing SQL Server and copying data into delta table on the lakehouse.
- Uploading files from your computer.
- Copying and merging multiple tables from other alehouses into a new delta table.
- Connecting to a streaming source to land data in a lakehouse.
- Referencing data without copying it from other internal lakehouses or external sources.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

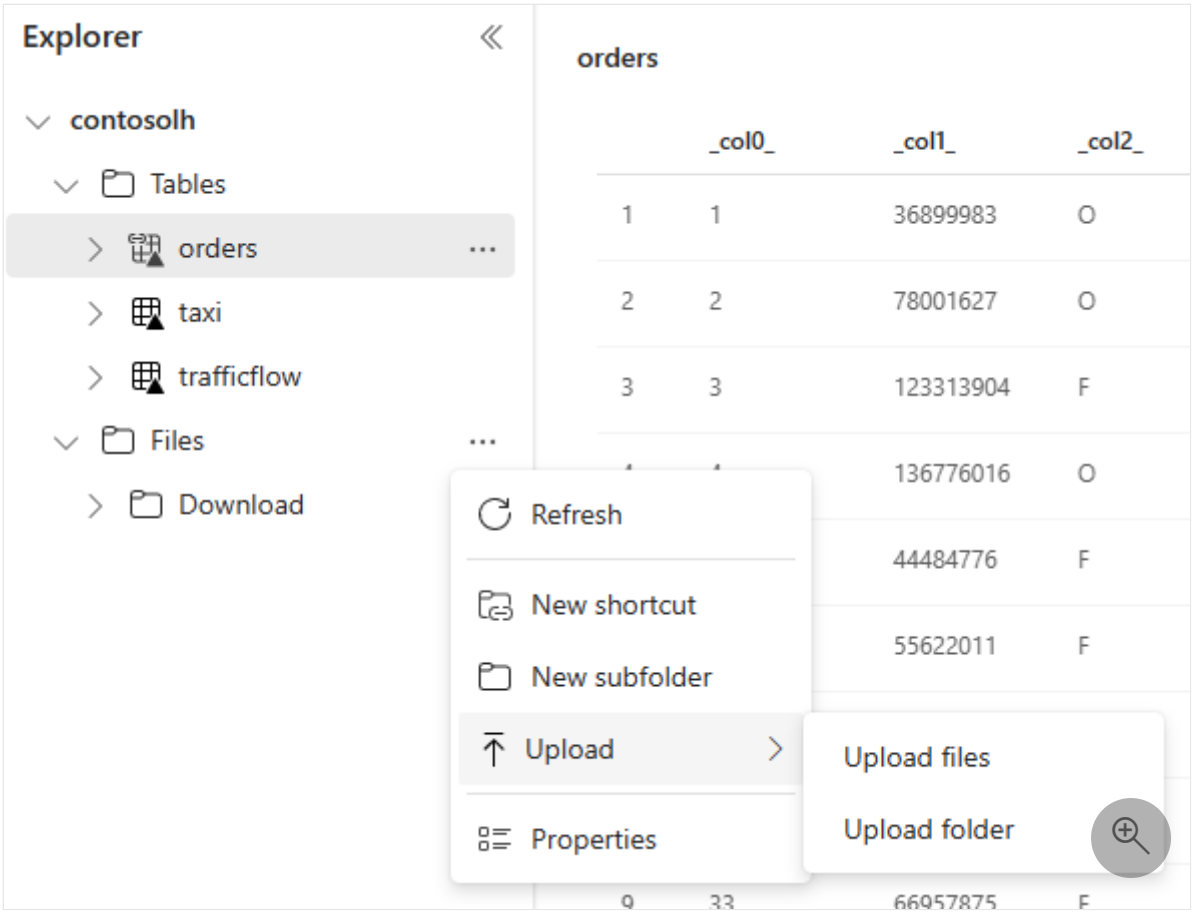
Different ways to load data in lakehouse

In Microsoft Fabric, there are a few ways you can get data into a lakehouse:

- File upload from local computer.
- Run a copy tool in pipelines.
- Set up a dataflow.
- Apache Spark libraries in notebook code

Local file upload

You can also upload data stored on your local machine. You can do it directly in the lakehouse explorer.



Copy tool in pipelines

The Copy tool is a highly scalable Data Integration solution that allows you to connect to different data sources and load the data either in original format or convert it to a delta table. Copy tool is a part of pipelines activities that can be orchestrated in multiple ways, such as scheduling or triggering based on event. See, How to copy data using copy activity.

Dataflows

For users that are familiar to Power BI dataflows same tool is available to land data in Lakehouse. You can quickly access it from Lakehouse explorer "Get data" option, and land data from over 200 connectors. See, Create your first dataflow to get and transform data.

Notebook code

You can use available Spark libraries to connect to a data source directly, load data to data frame and then save it in a lakehouse. It's the most open way to load data in the lakehouse that user code is fully managing.

> **Note**
>
> External delta tables created with Spark code won't be visible to SQL endpoint. Use shortcuts in Table space to make external delta tables visible for SQL endpoint.

Considerations when choosing approach to load data

Use case	Recommendation
Small file upload from local machine	Use Local file upload
Small data or specific connector	Use Dataflows
Large data source	Use Copy tool in pipelines
Complex data transformations	Use Notebook code

Next steps

- Overview: How to use notebook together with lakehouse
- Quickstart: Create your first pipeline to copy data.
- How to: How to copy data using Copy activity in Data pipeline.
- Tutorial: Move data into lakehouse via Copy assistant.

What are shortcuts in lakehouse?

Article • 05/23/2023

Shortcuts in a lakehouse allow users to reference data without copying it. It unifies data from different lakehouses, workspaces, or external storage, such as ADLS Gen2 or AWS S3. You can quickly make large amounts of data available in your lakehouse locally without the latency of copying data from the source.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Setting up a shortcut

To create a shortcut, open Lakehouse Explorer and select where to place the shortcut under Tables or Files. Creating a shortcut to Delta formatted table under Tables in Lakehouse Explorer will automatically register it as a table, enabling data access through Spark, SQL endpoint, and default dataset. Spark can access shortcuts in Files for data science projects or for transformation into structured data.

Access Control for shortcuts

Shortcuts to Microsoft Fabric internal sources will use the calling user identity. External shortcuts will use connectivity details, including security details specified when the shortcut is created.

Next steps

- Learn more about shortcuts

Get streaming data into lakehouse with Spark structured streaming

Article • 05/23/2023

Structured Streaming is a scalable and fault-tolerant stream processing engine built on Spark. Spark takes care of running the streaming operation incrementally and continuously as data continues to arrive.

Structured streaming became available in Spark 2.2. Since then, it has been the recommended approach for data streaming. The fundamental principle behind structured stream is to treat a live data stream as a table where new data is always continuously appended, like a new row in a table. There are a few defined built-in streaming file sources such as CSV, JSON, ORC, Parquet and built-in support for messaging services like Kafka and Event Hubs.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

This article provides insights into how to optimize the processing and ingestion of events through Spark structure streaming in production environments with high throughput. The suggested approaches include:

- Data streaming throughput optimization
- Optimizing write operations in the delta table and
- Event batching

Spark job definitions and Spark notebooks

Spark notebooks are an excellent tool for validating ideas and doing experiments to get insights from your data or code. Notebooks are widely used in data preparation, visualization, machine learning, and other big data scenarios. Spark job definitions are non-interactive code-oriented tasks running on a Spark cluster for long periods. Spark job definitions provide robustness and availability.

Spark notebooks are excellent source to test the logic of your code and address all the business requirements. However to keep it running in a production scenario, Spark job

definitions with Retry Policy enabled are the best solution.

Retry policy for Spark Job Definitions

In Microsoft Fabric, the user can set a retry policy for Spark Job Definition jobs. Though the script in the job might be infinite, the infrastructure running the script might incur an issue requiring stopping the job. Or the job could be eliminated due to underlying infrastructure patching needs. The retry policy allows the user to set rules for automatically restarting the job if it stops because of any underlying issues. The parameters specify how often the job should be restarted, up to infinite retries, and setting time between retries. That way, the users can ensure that their Spark Job Definition jobs continue running infinitely until the user decides to stop them.

Streaming sources

Setting up streaming with Event Hubs require basic configuration, which, includes Event Hubs namespace name, hub name, shared access key name, and the consumer group. A consumer group is a view of an entire event hub. It enables multiple consuming applications to have a separate view of the event stream and to read the stream independently at their own pace and with their offsets.

Partitions are an essential part of being able to handle a high volume of data. A single processor has a limited capacity for handling events per second, while multiple processors can do a better job when executed in parallel. Partitions allow the possibility of processing large volumes of events in parallel.

If too many partitions are used with a low ingestion rate, partition readers deal with a tiny portion of this data, causing nonoptimal processing. The ideal number of partitions directly depends on the desired processing rate. If you want to scale your event processing, consider adding more partitions. There's no specific throughput limit on a partition. However, the aggregate throughput in your namespace is limited by the number of throughput units. As you increase the number of throughput units in your namespace, you may want extra partitions to allow concurrent readers to achieve their maximum throughput.

The recommendation is to investigate and test the best number of partitions for your throughput scenario. But it's common to see scenarios with high throughput using 32 or more partitions.

Azure Event Hubs Connector for Apache Spark (link) is recommended to connect Spark application to Azure Event Hubs.

Lakehouse as streaming sink

Delta Lake is an open-source storage layer that provides ACID (atomicity, consistency, isolation, and durability) transactions on top of data lake storage solutions. Delta Lake also supports scalable metadata handling, schema evolution, time travel (data versioning), open format, and other features.

In Fabric Data Engineering, Delta Lake is used to:

- Easily upsert (insert/update) and delete data using Spark SQL.
- Compact data to minimize the time spent querying data.
- View the state of tables before and after operations are executed.
- Retrieve a history of operations performed on tables.

Delta is added as one of the possible outputs sinks formats used in writeStream – more information about the existing output sinks can be found here.

The following example demonstrates how it's possible to stream data into Delta Lake.

PySpark

```
import pyspark.sql.functions as f
from pyspark.sql.types import *

df = spark \
  .readStream \
  .format("eventhubs") \
  .options(**ehConf) \
  .load()

Schema = StructType([StructField("<column_name_01>", StringType(), False),
                     StructField("<column_name_02>", StringType(), False),
                     StructField("<column_name_03>", DoubleType(), True),
                     StructField("<column_name_04>", LongType(), True),
                     StructField("<column_name_05>", LongType(), True)])

rawData = df \
  .withColumn("bodyAsString", f.col("body").cast("string")) \
  .select(from_json("bodyAsString", Schema).alias("events")) \
  .select("events.*") \
  .writeStream \
  .format("delta") \
  .option("checkpointLocation", " Files/checkpoint") \
  .outputMode("append") \
  .toTable("deltaeventstable")
```

About the code snipped in the example:

- *format()* is the instruction that defines the output format of the data.

- *outputMode()* defines in which way the new rows in the streaming are written (that is, append, overwrite).
- *toTable()* persists the streamed data into a Delta table created using the value passed as parameter.

Optimizing Delta writes

Data partitioning is a critical part in creating a robust streaming solution: partitioning improves the way data is organized, and it also improves the throughput. Files easily get fragmented after Delta operations, resulting in too many small files. And too large files are also a problem, due to the long time to write them on the disk. The challenge with data partitioning is finding the proper balance that results in optimal file sizes. Spark supports partitioning in memory and on disk. Properly partitioned data can provide the best performance when persisting data to Delta Lake and querying data from Delta Lake.

- When partitioning data on disk, you can choose how to partition the data based on columns by using *partitionBy()*. *partitionBy()* is a function used to partition large dataset into smaller files based on one or multiple columns provided while writing to disk. Partitioning is a way to improve the performance of query when working with a large dataset. Avoid choosing a column that generates too small or too large partitions. Define a partition based on a set of columns with a good cardinality and split the data into files of optimal size.
- Partitioning data in memory can be done using *repartition()* or *coalesce()* transformations, distributing data on multiple worker nodes and creating multiple tasks that can read and process data in parallel using the fundamentals of Resilient Distributed Dataset (RDD). It allows dividing dataset into logical partitions, which can be computed on different nodes of the cluster.
  - *repartition()* is used to increase or decrease the number of partitions in memory. Repartition reshuffles whole data over the network and balances it across all partitions.
  - *coalesce()* is only used to decrease the number of partitions efficiently. That is an optimized version of *repartition()* where the movement of data across all partitions is lower using coalesce().

Combining both partitioning approaches is a good solution in scenario with high throughput. *repartition()* creates a specific number of partitions in memory, while *partitionBy()* writes files to disk for each memory partition and partitioning column. The following example illustrates the usage of both partitioning strategies in the same Spark job: data is first split into 48 partitions in memory (assuming we have total 48 CPU cores), and then partitioned on disk based in two existing columns in the payload.

PySpark

```
import pyspark.sql.functions as f
from pyspark.sql.types import *
import json

rawData = df \
  .withColumn("bodyAsString", f.col("body").cast("string")) \
  .select(from_json("bodyAsString", Schema).alias("events")) \
  .select("events.*") \
  .repartition(48) \
  .writeStream \
  .format("delta") \
  .option("checkpointLocation", " Files/checkpoint") \
  .outputMode("append") \
  .partitionBy("<column_name_01>", "<column_name_02>") \
  .toTable("deltaeventstable")
```

Optimized Write

Another option to optimize writes to Delta Lake is using Optimized Write. Optimized Write is an optional feature that improves the way data is written to Delta table. Spark merges or splits the partitions before writing the data, maximizing the throughput of data being written to the disk. However, it incurs full shuffle, so for some workloads it can cause a performance degradation. Jobs using *coalesce()* and/or *repartition()* to partition data on disk can be refactored to start using Optimized Write instead.

The following code is an example of the use of Optimized Write. Note that *partitionBy()* is still used.

PySpark

```
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled", true)

rawData = df \
 .withColumn("bodyAsString", f.col("body").cast("string")) \
  .select(from_json("bodyAsString", Schema).alias("events")) \
  .select("events.*") \
  .writeStream \
  .format("delta") \
  .option("checkpointLocation", " Files/checkpoint") \
  .outputMode("append") \
  .partitionBy("<column_name_01>", "<column_name_02>") \
  .toTable("deltaeventstable")
```

Batching events

In order to minimize the number of operations to improve the time spent ingesting data into Delta lake, batching events is a practical alternative.

Triggers define how often a streaming query should be executed (triggered) and emit a new data, setting them up defines a periodical processing time interval for microbatches, accumulating data and batching events into few persisting operations, instead of writing into disk all the time.

The following example shows a streaming query where events are periodically processed in intervals of one minute.

PySpark

```
rawData = df \
  .withColumn("bodyAsString", f.col("body").cast("string")) \
  .select(from_json("bodyAsString", Schema).alias("events")) \
  .select("events.*") \
  .repartition(48) \
  .writeStream \
  .format("delta") \
  .option("checkpointLocation", " Files/checkpoint") \
  .outputMode("append") \
  .partitionBy("<column_name_01>", "<column_name_02>") \
  .trigger(processingTime="1 minute") \
  .toTable("deltaeventstable")
```

The advantage of combining batching of events in Delta table writing operations is that it creates larger Delta files with more data in them, avoiding small files. You should analyze the amount of data being ingested and find the best processing time to optimize the size of the Parquet files created by Delta library.

Monitoring

Spark 3.1 and higher versions have a built-in structured streaming UI (link) containing the following streaming metrics:

- Input Rate
- Process Rate
- Input Rows
- Batch Duration
- Operation Duration

Next steps

- Get streaming data into lakehouse and access with SQL endpoint.

Workspace roles in Lakehouse

Article • 05/23/2023

Workspace roles define what user can do with Microsoft Fabric items. Roles can be assigned to individuals or security groups from workspace view. See, Give users access to workspaces.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

The user can be assigned to the following roles:

- Admin
- Member
- Contributor
- Viewer

In a lakehouse the users with Admin, Member, and Contributor roles can perform all CRUD operations on all data. A user with Viewer role can only read data stored in Tables using SQL endpoint.

> **Important**
>
> When accessing data using SQL endpoint with Viewer role, **make sure SQL access policy is granted to read required tables**.

Next steps

- Roles in workspaces

What is Spark compute in Microsoft Fabric?

Article • 05/23/2023

Applies to: ✅ Data Engineering and Data Science in Microsoft Fabric

Microsoft Fabric Data Engineering and Data Science experiences operate on a fully managed Spark compute platform. This platform is designed to deliver unparalleled speed and efficiency. With starter pools, you can expect rapid spark session initialization, typically within 5 to 10 seconds. It eliminates the need for manual setup. Furthermore, you also get the flexibility to customize Spark pools according to the specific data engineering and data science requirements. It enables an optimized and tailored analytics experience.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

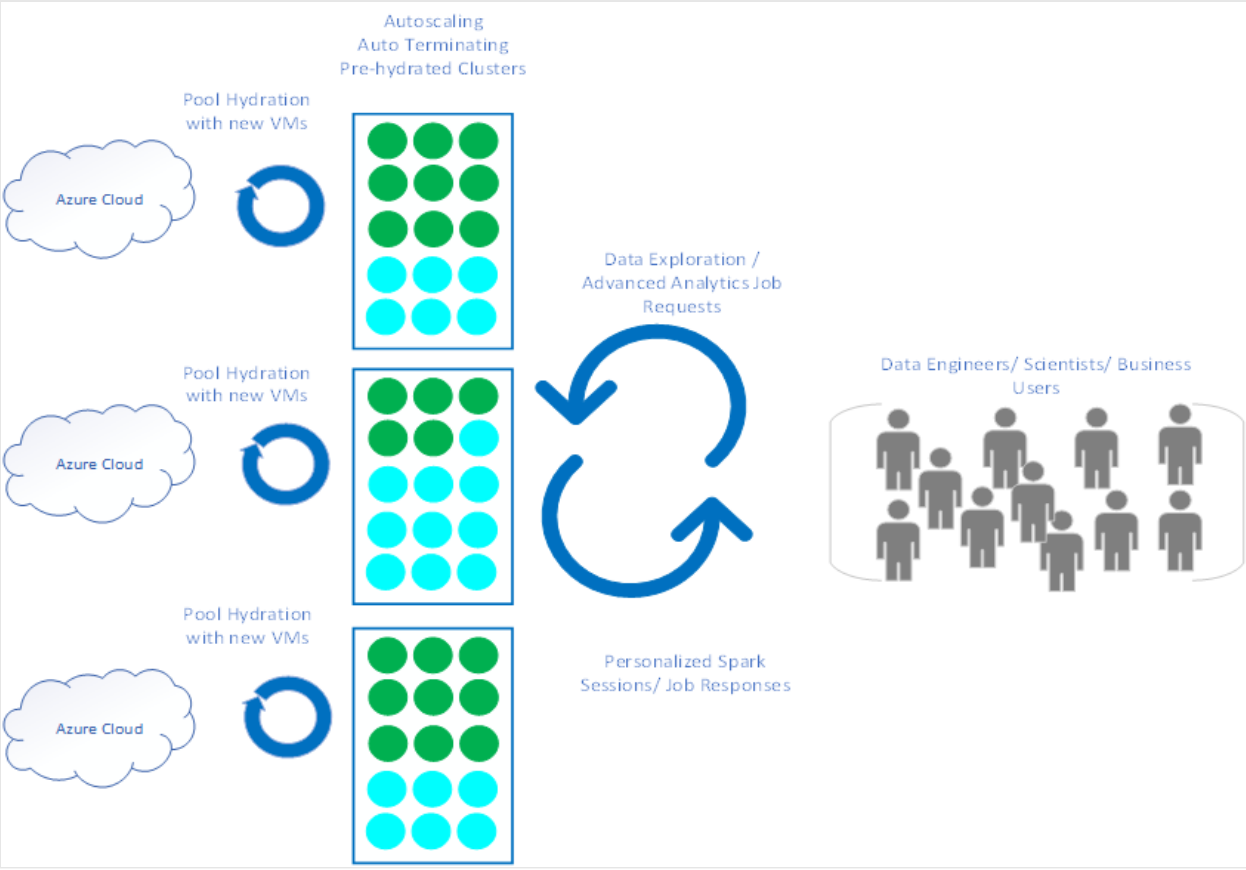
Starter pools

Starter pools are a fast and easy way to use Spark on the Microsoft Fabric platform within seconds. You can use Spark sessions right away, instead of waiting for Spark to set up the nodes for you. This helps you do more with data and get insights quicker.

Starter Pool Configuration

Node family	Memory optimized
Node Size	Medium
Min and Max Nodes	[1, 10]
Autoscale	On
Dynamic Allocation	On

Starter pools have Spark clusters that are always on and ready for your requests. They use medium nodes that will dynamically scale-up based on your Spark job needs.



Starter pools also have default settings that let you install libraries quickly without slowing down the session start time. However, if you want to use extra custom Spark properties or libraries from your workspace or capacity settings, it may take longer for

Spark to get the nodes for you. You only pay for starter pools when you are using Spark sessions to run queries. You don't pay for the time when Spark is keeping the nodes ready for you.

Spark pools

A Spark pool is a way of telling Spark what kind of resources you need for your data analysis tasks. You can give your Spark pool a name, and choose how many and how big the nodes (the machines that do the work) are. You can also tell Spark how to adjust the number of nodes depending on how much work you have. Creating a Spark pool is free; you only pay when you run a Spark job on the pool, and then Spark will set up the nodes for you.

If you don't use your Spark pool for 2 minutes after your job is done, Spark will automatically delete it. This is called the "time to live" property, and you can change it if you want. If you are a workspace admin, you can also create custom Spark pools for your workspace, and make them the default option for other users. This way, you can save time and avoid setting up a new Spark pool every time you run a notebook or a Spark job. Custom Spark pools take about 3 minutes to start, because Spark has to get the nodes from Azure.

The size and number of nodes you can have in your custom Spark pool depends on how much capacity you have in your Microsoft Fabric capacity. This is a measure of how much computing power you can use in Azure. One way to think of it is that two Spark VCores (a unit of computing power for Spark) equals one capacity unit. For example, if you have a Fabric capacity SKU F64, that means you have 64 capacity units, which is equivalent to 128 Spark VCores. You can use these Spark VCores to create nodes of different sizes for your custom Spark pool, as long as the total number of Spark VCores does not exceed 128.

Possible custom pool configurations for F64 based on the above example

Fabric Capacity SKU	Capacity Units	Spark VCores	Node Size	Max Number of Nodes
F64	64	128	Small	32
F64	64	128	Medium	16
F64	64	128	Large	8
F64	64	128	X-Large	4
F64	64	128	XX-Large	2

> **Note**
>
> To create custom pools, you should have the **admin** permissions for the workspace. And the Microsoft Fabric capacity admin should have granted permissions to allow workspace admins to size their custom spark pools. To learn more, see **Get Started with Custom Spark Pools in Fabric**

Nodes

Apache Spark pool instance consists of one head node and two or more worker nodes with a minimum of three nodes in a Spark instance. The head node runs extra management services such as Livy, Yarn Resource Manager, Zookeeper, and the Spark driver. All nodes run services such as Node Agent and Yarn Node Manager. All worker nodes run the Spark Executor service.

Node sizes

A Spark pool can be defined with node sizes that range from a small compute node with 4 vCore and 32 GB of memory to a large compute node with 64 vCore and 512 GB of memory per node. Node sizes can be altered after pool creation although the active session would have to be restarted.

Size	vCore	Memory
Small	4	32 GB
Medium	8	64 GB
Large	16	128 GB
X-Large	32	256 GB
XX-Large	64	512 GB

Autoscale

Autoscale for Apache Spark pools allows automatic scale up and down of compute resources based on the amount of activity. When the autoscale feature is enabled, you set the minimum, and maximum number of nodes to scale. When the autoscale feature is disabled, the number of nodes set remains fixed. This setting can be altered after pool creation although the instance may need to be restarted.

Dynamic allocation

Dynamic allocation allows the spark application to request more executors if the tasks exceed the load that current executors can bear. It also releases the executors when the jobs are completed and if the spark application is moving to idle state. Enterprise users often find it hard to tune the executor configurations. Because they're vastly different across different stages of a Spark Job execution process. These are also dependent on the volume of data processed which changes from time to time. Users can enable dynamic allocation of executors option as part of the pool configuration, which would enable automatic allocation of executors to the spark application based on the nodes available in the Spark pool.

When dynamic allocation option is enabled, for every spark application submitted. The system reserves executors during the job submission step based on the maximum nodes, which were specified by the user to support successful auto scale scenarios.

Next steps

- Get Started with Data Engineering/Science Admin Settings for your Fabric Capacity
- Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace

Apache Spark Runtime in Fabric

Article • 05/23/2023

The Microsoft Fabric Runtime is an Azure-integrated platform based on Apache Spark that enables the execution and management of data engineering and data science experiences. It combines key components from both internal and open-source sources, providing customers with a comprehensive solution. For simplicity, we refer to the Microsoft Fabric Runtime powered by Apache Spark as Fabric Runtime.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

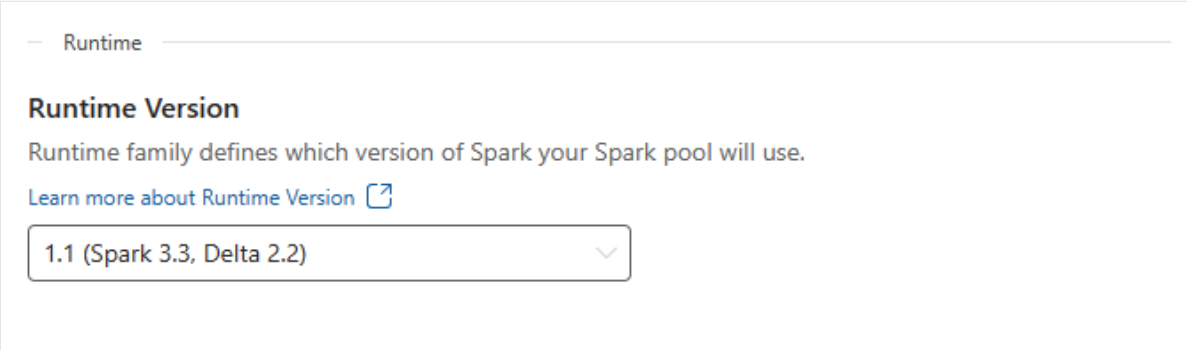
Major components of the Fabric Runtime:

- **Apache Spark** - a powerful open-source distributed computing library, to enable large-scale data processing and analytics tasks. Apache Spark provides a versatile and high-performance platform for data engineering and data science experiences.
- **Delta Lake** - an open-source storage layer that brings ACID transactions and other data reliability features to Apache Spark. Integrated within the Microsoft Fabric Runtime, Delta Lake enhances the data processing capabilities and ensures data consistency across multiple concurrent operations.
- **Default-level Packages for Java/Scala, Python, and R** to support diverse programming languages and environments. These packages are automatically installed and configured, allowing developers to apply their preferred programming languages for data processing tasks.
- The Microsoft Fabric Runtime is built upon **a robust open-source operating system (Ubuntu)**, ensuring compatibility with various hardware configurations and system requirements.

Runtime 1.1

Microsoft Fabric Runtime 1.1 is the default and currently the only runtime offered within the Microsoft Fabric platform. The Runtime 1.1 major components are:

- Operating System: Ubuntu 18.04
- Java: 1.8.0_282
- Scala: 2.12.15
- Python: 3.10
- Delta Lake: 2.2
- R: 4.2.2



Microsoft Fabric Runtime 1.1 comes with a collection of default level packages, including a full Anaconda installation and commonly used libraries for Java/Scala, Python, and R. These libraries are automatically included when using notebooks or jobs in the Microsoft Fabric platform. Refer to the documentation for a complete list of libraries.

Microsoft Fabric periodically rolls out maintenance updates for Runtime 1.1, providing bug fixes, performance enhancements, and security patches. *Staying up to date ensures optimal performance and reliability for your data processing tasks.*

New features and improvements

Apache Spark 3.3.1

Following is an extended summary of key new features related to Apache Spark version 3.3.0 and 3.3.1:

- **Row-level filtering**: improve the performance of joins by prefiltering one side as long as there are no deprecation or regression impacts.oin using a Bloom filter and IN predicate generated from the values from the other side of the join (SPARK-32268)
- Improve the compatibility of Spark with the SQL standard:**ANSI enhancements** (SPARK-38860)
- Error Message Improvements to identify problems faster and take the necessary steps to resolve it (SPARK-38781)
- Support **complex types for Parquet vectorized reader.** Previously, Parquet vectorized reader hasn't supported nested column type (struct, array, and map). The Apache Spark 3.3 contains an implementation of nested column vectorized reader for FB-ORC in our internal fork of Spark. It impacts performance improvement compared to nonvectorized reader when reading nested columns. In addition, this implementation can also help improve the non-nested column performance when reading non-nested and nested columns together in one query (SPARK-34863)
- Allows users to query the metadata of the input files for all file formats, expose them as **built-in hidden columns** meaning **users can only see them when they explicitly reference them** (for example, file path and file name) (SPARK-37273)
- Provide a profiler for Python/Pandas UDFs (SPARK-37443)

- Previously, streaming queries with Trigger, which was loading all of the available data in a single batch. Because of this, the amount of data the queries could process was limited, or the Spark driver would be out of memory. Now, introducing **Trigger.AvailableNow** for running streaming queries like Trigger once in multiple batches (SPARK-36533)
- More comprehensive DS V2 push down capabilities (SPARK-38788)
- Executor **Rolling in Kubernetes** environment (SPARK-37810)
- Support **Customized Kubernetes** Schedulers (SPARK-36057)
- Migrating from **log4j 1 to log4j 2** (SPARK-37814) to gain in:
  - Performance: Log4j 2 is faster than Log4j 1. Log4j 2 uses **asynchronous logging by default,** which can improve performance significantly.
  - Flexibility: Log4j 2 provides more flexibility in terms of configuration. It supports **multiple configuration formats**, including XML, JSON, and YAML.
  - Extensibility: Log4j 2 is designed to be extensible. It allows developers to **create custom plugins and appenders** to extend the functionality of the logging framework.
  - Security: Log4j 2 provides better security features than Log4j 1. It supports **encryption and secure socket layers** for secure communication between applications.
  - Simplicity: Log4j 2 is simpler to use than Log4j 1. It has **a more intuitive API** and a simpler configuration process.
- Introduce shuffle on **SinglePartition** to improve parallelism and fix performance regression for joins in Spark 3.3 vs Spark 3.2 (SPARK-40703)
- Optimize **TransposeWindow** rule to extend applicable cases and optimize time complexity (SPARK-38034)
- To have a parity in doing TimeTravel via SQL and Dataframe option, **support timestamp** in seconds for **TimeTravel** using Dataframe options (SPARK-39633])
- Optimize **global Sort** to **RepartitionByExpression** to save a local sort (SPARK-39911)
- Ensure the **output partitioning** is user-specified in **AQE** (SPARK-39915)
- Update Parquet V2 columnar check for nested fields (SPARK-39951)
- Reading in **a parquet file partitioned on disk by a `Byte`-type column** (SPARK-40212)
- Fix column pruning in CSV when _corrupt_record is selected (SPARK-40468)

Delta Lake 2.2

The key features in this release are as follows:

- LIMIT pushdown into Delta scan. Improve the performance of queries containing `LIMIT` clauses by pushing down the `LIMIT` into Delta scan during query planning. Delta scan uses the `LIMIT` and the file-level row counts to reduce the number of files scanned which helps the queries read far less number of files and could make `LIMIT` queries faster by 10-100x depending upon the table size.
- Aggregate pushdown into Delta scan for SELECT COUNT(*). Aggregation queries such as `SELECT COUNT(*)` on Delta tables are satisfied using file-level row counts in Delta table metadata rather than counting rows in the underlying data files. This significantly reduces the query time as the query just needs to read the table metadata and could make full table count queries faster by 10-100x.
- Support for collecting file level statistics as part of the CONVERT TO DELTA command. These statistics potentially help speed up queries on the Delta table. By default the statistics are collected now as part of the CONVERT TO DELTA command. In order to disable statistics collection, specify `NO STATISTICS` clause in the command. Example: `CONVERT TO DELTA table_name NO STATISTICS`
- Improve performance of the DELETE command by pruning the columns to read when searching for files to rewrite.
- Fix for a bug in the DynamoDB-based S3 multi-cluster mode configuration. The previous version wrote an incorrect timestamp, which was used by DynamoDB's TTL feature to clean up expired items. This timestamp value has been fixed and the table attribute renamed from `commitTime` to `expireTime`. If you already have TTL enabled, follow the migration steps here.
- Fix nondeterministic behavior during MERGE when working with sources that are nondeterministic.
- Remove the restrictions for using Delta tables with column mapping in certain Streaming + CDF cases. Earlier we used to block Streaming+CDF if the Delta table has column mapping enabled even though it doesn't contain any RENAME or DROP columns.
- Improve the monitoring of the Delta state construction queries (other queries run as part of planning) by making them visible in the Spark UI.
- Support for multiple `where()` calls in Optimize scala/python API
- Support for passing Hadoop configurations via DeltaTable API
- Support partition column names starting with `.` or `_` in CONVERT TO DELTA command.
- Improvements to metrics in table history
  - Fix a metric in MERGE command
  - Source type metric for CONVERT TO DELTA
  - Metrics for DELETE on partitions
  - More vacuum stats

- Fix for accidental protocol downgrades with RESTORE command. Until now, RESTORE TABLE may downgrade the protocol version of the table, which could have resulted in inconsistent reads with time travel. With this fix, the protocol version is never downgraded from the current one.
- Fix a bug in `MERGE INTO` when there are multiple `UPDATE` clauses and one of the UPDATEs is with a schema evolution.
- Fix a bug where sometimes active `SparkSession` object isn't found when using Delta APIs
- Fix an issue where partition schema couldn't be set during the initial commit.
- Catch exceptions when writing `last_checkpoint` file fails.
- Fix an issue when restarting a streaming query with `AvailableNow` trigger on a Delta table.
- Fix an issue with CDF and Streaming where the offset isn't correctly updated when there are no data changes

Check the source and full release notes here.

Default level packages for Java/Scala libraries

Below you can find the table with listing all the default level packages for Java/Scala and their respective versions.

GroupId	ArtifactId	Version
com.aliyun	aliyun-java-sdk-core	4.5.10
com.aliyun	aliyun-java-sdk-kms	2.11.0
com.aliyun	aliyun-java-sdk-ram	3.1.0
com.aliyun	aliyun-sdk-oss	3.13.0
com.amazonaws	aws-java-sdk-bundle	1.11.1026
com.chuusai	shapeless_2.12	2.3.7
com.esotericsoftware	kryo-shaded	4.0.2
com.esotericsoftware	minlog	1.3.0
com.fasterxml.jackson	jackson-annotations-2.13.4.jar	
com.fasterxml.jackson	jackson-core	2.13.4
com.fasterxml.jackson	jackson-core-asl	1.9.13
com.fasterxml.jackson	jackson-databind	2.13.4.1
com.fasterxml.jackson	jackson-dataformat-cbor	2.13.4
com.fasterxml.jackson	jackson-mapper-asl	1.9.13

GroupId	ArtifactId	Version
com.fasterxml.jackson	jackson-module-scala_2.12	2.13.4
com.github.joshelser	dropwizard-metrics-hadoop-metrics2-reporter	0.1.2
com.github.wendykierp	JTransforms	3.1
com.google.code.findbugs	jsr305	3.0.0
com.google.code.gson	gson	2.8.6
com.google.flatbuffers	flatbuffers-java	1.12.0
com.google.guava	guava	14.0.1
com.google.protobuf	protobuf-java	2.5.0
com.googlecode.json-simple	json-simple	1.1.1
com.jcraft	jsch	0.1.54
com.jolbox	bonecp	0.8.0.RELEASE
com.linkedin.isolation-forest	isolation-forest_3.2.0_2.12	2.0.8
com.ning	compress-lzf	1.1
com.qcloud	cos_api-bundle	5.6.19
com.sun.istack	istack-commons-runtime	3.0.8
com.tdunning	json	1.8
com.thoughtworks.paranamer	paranamer	2.8
com.twitter	chill-java	0.10.0
com.twitter	chill_2.12	0.10.0
com.typesafe	config	1.3.4
com.zaxxer	HikariCP	2.5.1
commons-cli	commons-cli	1.5.0
commons-codec	commons-codec	1.15
commons-collections	commons-collections	3.2.2
commons-dbcp	commons-dbcp	1.4
commons-io	commons-io	2.11.0
commons-lang	commons-lang	2.6
commons-logging	commons-logging	1.1.3
commons-pool	commons-pool	1.5.4.jar

GroupId	ArtifactId	Version
dev.ludovic.netlib	arpack	2.2.1
dev.ludovic.netlib	blas	2.2.1
dev.ludovic.netlib	lapack	2.2.1
io.airlift	aircompressor	0.21
io.dropwizard.metrics	metrics-core	4.2.7
io.dropwizard.metrics	metrics-graphite	4.2.7
io.dropwizard.metrics	metrics-jmx	4.2.7
io.dropwizard.metrics	metrics-json	4.2.7
io.dropwizard.metrics	metrics-jvm	4.2.7
io.netty	netty-all	4.1.74.Final
io.netty	netty-buffer	4.1.74.Final
io.netty	netty-codec	4.1.74.Final
io.netty	netty-common	4.1.74.Final
io.netty	netty-handler	4.1.74.Final
io.netty	netty-resolver	4.1.74.Final
io.netty	netty-tcnative-classes	2.0.48.Final
io.netty	netty-transport	4.1.74.Final
io.netty	netty-transport-classes-epoll	4.1.74.Final
io.netty	netty-transport-classes-kqueue	4.1.74.Final
io.netty	netty-transport-native-epoll	4.1.74.Final-linux-aarch_64
io.netty	netty-transport-native-epoll	4.1.74.Final-linux-x86_64
io.netty	netty-transport-native-kqueue	4.1.74.Final-osx-aarch_64
io.netty	netty-transport-native-kqueue	4.1.74.Final-osx-x86_64
io.netty	netty-transport-native-unix-common	4.1.74.Final
io.opentracing	opentracing-api	0.33.0
io.opentracing	opentracing-noop	0.33.0
io.opentracing	opentracing-util	0.33.0
jakarta.annotation	jakarta.annotation-api	1.3.5
jakarta.inject	jakarta.inject	2.6.1

GroupId	ArtifactId	Version
jakarta.servlet	jakarta.servlet-api	4.0.3
jakarta.validation-api	2.0.2	
jakarta.ws.rs	jakarta.ws.rs-api	2.1.6
jakarta.xml.bind	jakarta.xml.bind-api	2.3.2
javax.activation	activation	1.1.1
javax.jdo	jdo-api	3.0.1
javax.transaction	jta	1.1
javax.xml.bind	jaxb-api	2.2.11
javolution	javolution	5.5.1
jline	jline	2.14.6
joda-time	joda-time	2.10.13
net.razorvine	pickle	1.2
net.sf.jpam	jpam	1.1
net.sf.opencsv	opencsv	2.3
net.sf.py4j	py4j	0.10.9.5
net.sourceforge.f2j	arpack_combined_all	0.1
org.antlr	ST4	4.0.4
org.antlr	antlr-runtime	3.5.2
org.antlr	antlr4-runtime	4.8
org.apache.arrow	arrow-format	7.0.0
org.apache.arrow	arrow-memory-core	7.0.0
org.apache.arrow	arrow-memory-netty	7.0.0
org.apache.arrow	arrow-vector	7.0.0
org.apache.avro	avro	1.11.0
org.apache.avro	avro-ipc	1.11.0
org.apache.avro	avro-mapred	1.11.0
org.apache.commons	commons-collections4	4.4
org.apache.commons	commons-compress	1.21
org.apache.commons	commons-crypto	1.1.0
org.apache.commons	commons-lang3	3.12.0

GroupId	ArtifactId	Version
org.apache.commons	commons-math3	3.6.1
org.apache.commons	commons-pool2	2.11.1
org.apache.commons	commons-text	1.10.0
org.apache.curator	curator-client	2.13.0
org.apache.curator	curator-framework	2.13.0
org.apache.curator	curator-recipes	2.13.0
org.apache.derby	derby	10.14.2.0
org.apache.hadoop	hadoop-aliyun	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-annotations	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-aws	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-azure	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-azure-datalake	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-client-api	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-client-runtime	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-cloud-storage	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-cos	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-openstack	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-shaded-guava	1.1.1
org.apache.hadoop	hadoop-yarn-server-web-proxy	3.3.3.5.2-90111858
org.apache.hive	hive-beeline	2.3.9
org.apache.hive	hive-cli	2.3.9
org.apache.hive	hive-common	2.3.9
org.apache.hive	hive-exec	2.3.9
org.apache.hive	hive-jdbc	2.3.9
org.apache.hive	hive-llap-common	2.3.9
org.apache.hive	hive-metastore	2.3.9
org.apache.hive	hive-serde	2.3.9
org.apache.hive	hive-service-rpc	3.1.2
org.apache.hive	hive-shims-0.23	2.3.9
org.apache.hive	hive-shims	2.3.9

GroupId	ArtifactId	Version
org.apache.hive	hive-shims-common	2.3.9
org.apache.hive	hive-shims-scheduler	2.3.9
org.apache.hive	hive-storage-api	2.7.2
org.apache.hive	hive-vector-code-gen	2.3.9
org.apache.httpcomponents	httpclient	4.5.13
org.apache.httpcomponents	httpcore	4.4.14
org.apache.httpcomponents	httpmime	4.5.13
org.apache.httpcomponents.client5	httpclient5	5.1.3
org.apache.ivy	ivy	2.5.1
org.apache.kafka	kafka-clients	2.8.1
org.apache.logging.log4j	log4j-1.2-api	2.17.2
org.apache.logging.log4j	log4j-api	2.17.2
org.apache.logging.log4j	log4j-core	2.17.2
org.apache.logging.log4j	log4j-slf4j-impl	2.17.2
org.apache.orc	orc-core	1.7.6
org.apache.orc	orc-mapreduce	1.7.6
org.apache.orc	orc-shims	1.7.6
org.apache.parquet	parquet-column	1.12.3
org.apache.parquet	parquet-common	1.12.3
org.apache.parquet	parquet-encoding	1.12.3
org.apache.parquet	parquet-format-structures	1.12.3
org.apache.parquet	parquet-hadoop	1.12.3
org.apache.parquet	parquet-jackson	1.12.3
org.apache.qpid	proton-j	0.33.8
org.apache.thrift	libfb303	0.9.3
org.apache.thrift	libthrift	0.12.0
org.apache.yetus	audience-annotations	0.5.0
org.apiguardian	apiguardian-api	1.1.0
org.codehaus.janino	commons-compiler	3.0.16
org.codehaus.janino	janino	3.0.16

GroupId	ArtifactId	Version
org.codehaus.jettison	jettison	1.1
org.datanucleus	datanucleus-api-jdo	4.2.4
org.datanucleus	datanucleus-core	4.1.17
org.datanucleus	datanucleus-rdbms	4.1.19
org.datanucleusjavax.jdo	3.2.0-m3	
org.eclipse.jdt	core	1.1.2
org.eclipse.jetty	jetty-util	9.4.48.v20220622
org.eclipse.jetty	jetty-util-ajax	9.4.48.v20220622
org.fusesource.leveldbjni	leveldbjni-all	1.8
org.glassfish.hk2	hk2-api	2.6.1
org.glassfish.hk2	hk2-locator	2.6.1
org.glassfish.hk2	hk2-utils	2.6.1
org.glassfish.hk2	osgi-resource-locator	1.0.3
org.glassfish.hk2.external	aopalliance-repackaged	2.6.1
org.glassfish.jaxb	jaxb-runtime	2.3.2
org.glassfish.jersey.containers	jersey-container-servlet	2.36
org.glassfish.jersey.containers	jersey-container-servlet-core	2.36
org.glassfish.jersey.core	jersey-client	2.36
org.glassfish.jersey.core	jersey-common	2.36
org.glassfish.jersey.core	jersey-server	2.36
org.glassfish.jersey.inject	jersey-hk2	2.36
org.ini4j	ini4j	0.5.4
org.javassist	javassist	3.25.0-GA
org.javatuples	javatuples	1.2
org.jdom	jdom2	2.0.6
org.jetbrains	annotations	17.0.0
org.jodd	jodd-core	3.5.2
org.json4s	json4s-ast_2.12	3.7.0-M11
org.json4s	json4s-core_2.12	3.7.0-M11
org.json4s	json4s-jackson_2.12	3.7.0-M11

GroupId	ArtifactId	Version
org.json4s	json4s-scalap_2.12	3.7.0-M11
org.junit.jupiter	junit-jupiter	5.5.2
org.junit.jupiter	junit-jupiter-api	5.5.2
org.junit.jupiter	junit-jupiter-engine	5.5.2
org.junit.jupiter	junit-jupiter-params	5.5.2
org.junit.platform	junit-platform-commons	1.5.2
org.junit.platform	junit-platform-engine	1.5.2
org.lz4	lz4-java	1.8.0
org.objenesis	objenesis	3.2
org.openpnp	opencv	3.2.0-1
org.opentest4j	opentest4j	1.2.0
org.postgresql	postgresql	42.2.9
org.roaringbitmap	RoaringBitmap	0.9.25
org.roaringbitmap	shims	0.9.25
org.rocksdb	rocksdbjni	6.20.3
org.scala-lang	scala-compiler	2.12.15
org.scala-lang	scala-library	2.12.15
org.scala-lang	scala-reflect	2.12.15
org.scala-lang.modules	scala-collection-compat_2.12	2.1.1
org.scala-lang.modules	scala-java8-compat_2.12	0.9.0
org.scala-lang.modules	scala-parser-combinators_2.12	1.1.2
org.scala-lang.modules	scala-xml_2.12	1.2.0
org.scalactic	scalactic_2.12	3.2.14
org.scalanlp	breeze-macros_2.12	1.2
org.scalanlp	breeze_2.12	1.2
org.slf4j	jcl-over-slf4j	1.7.32
org.slf4j	jul-to-slf4j	1.7.32
org.slf4j	slf4j-api	1.7.32
org.typelevel	algebra_2.12	2.0.1
org.typelevel	cats-kernel_2.12	2.1.1

GroupId	ArtifactId	Version
org.typelevel	spire-macros_2.12	0.17.0
org.typelevel	spire-platform_2.12	0.17.0
org.typelevel	spire-util_2.12	0.17.0
org.xerial.snappy	snappy-java	1.1.8.4
oro	oro	2.0.8
pl.edu.icm	JLargeArrays	1.5

Default level packages for Python libraries

Below you can find the table with listing all the default level packages for Python and their respective versions.

Library	Version	Library	Version	Library	Version
_libgcc_mutex	0.1	ipykernel	6.22.0	pickleshare	0.7.5
_openmp_mutex	4.5	ipython	8.9.0	pillow	9.4.0
_py-xgboost-mutex	2.0	ipywidgets	8.0.4	pip	23.0.1
absl-py	1.4.0	isodate	0.6.1	pixman	0.40.0
adal	1.2.7	itsdangerous	2.1.2	pkginfo	1.9.6
adlfs	2023.1.0	jack	1.9.22	pkgutil-resolve-name	1.3.10
aiohttp	3.8.4	jedi	0.18.2	platformdirs	3.2.0
aiosignal	1.3.1	jeepney	0.8.0	plotly	5.13.0
alsa-lib	1.2.8	jinja2	3.1.2	ply	3.11
anyio	3.6.2	jmespath	1.0.1	pooch	1.7.0
argcomplete	2.1.2	joblib	1.2.0	portalocker	2.7.0
argon2-cffi	21.3.0	jpeg	9e	pox	0.3.2
argon2-cffi-bindings	21.2.0	jsonpickle	2.2.0	ppft	1.7.6.6
arrow-cpp	11.0.0	jsonschema	4.17.3	prettytable	3.6.0
asttokens	2.2.1	jupyter_client	8.1.0	prometheus_client	0.16.0
astunparse	1.6.3	jupyter_core	5.3.0	prompt-toolkit	3.0.38
async-timeout	4.0.2	jupyter_events	0.6.3	protobuf	4.21.12
atk-1.0	2.38.0	jupyter_server	2.2.1	psutil	5.9.4

Library	Version	Library	Version	Library	Version
attr	2.5.1	jupyter_server_terminals	0.4.4	pthread-stubs	0.4
attrs	22.2.0	jupyterlab_pygments	0.2.2	ptyprocess	0.7.0
aws-c-auth	0.6.24	jupyterlab_widgets	3.0.7	pulseaudio	16.1
aws-c-cal	0.5.20	keras	2.11.0	pulseaudio-client	16.1
aws-c-common	0.8.11	keras-preprocessing	1.1.2	pulseaudio-daemon	16.1
aws-c-compression	0.2.16	keyutils	1.6.1	pure_eval	0.2.2
aws-c-event-stream	0.2.18	kiwisolver	1.4.4	py-xgboost	1.7.1
aws-c-http	0.7.4	knack	0.10.1	py4j	0.10.9.5
aws-c-io	0.13.17	krb5	1.20.1	pyarrow	11.0.0
aws-c-mqtt	0.8.6	lame	3.100	pyasn1	0.4.8
aws-c-s3	0.2.4	lcms2	2.15	pyasn1-modules	0.2.7
aws-c-sdkutils	0.1.7	ld_impl_linux-64	2.40	pycosat	0.6.4
aws-checksums	0.1.14	lerc	4.0.0	pycparser	2.21
aws-crt-cpp	0.19.7	liac-arff	2.5.0	pygments	2.14.0
aws-sdk-cpp	1.10.57	libabseil	20220623.0	pyjwt	2.6.0
azure-common	1.1.28	libaec	1.0.6	pynacl	1.5.0
azure-core	1.26.4	libarrow	11.0.0	pyodbc	4.0.35
azure-datalake-store	0.0.51	libblas	3.9.0	pyopenssl	23.1.1
azure-graphrbac	0.61.1	libbrotlicommon	1.0.9	pyparsing	3.0.9
azure-identity	1.12.0	libbrotlidec	1.0.9	pyperclip	1.8.2
azure-mgmt-authorization	3.0.0	libbrotlienc	1.0.9	pyqt	5.15.7
azure-mgmt-containerregistry	10.1.0	libcap	2.67	pyqt5-sip	12.11.0
azure-mgmt-core	1.4.0	libcblas	3.9.0	pyrsistent	0.19.3
azure-mgmt-keyvault	10.2.1	libclang	15.0.7	pysocks	1.7.1
azure-mgmt-resource	21.2.1	libclang13	15.0.7	pyspark	3.3.1
azure-mgmt-storage	20.1.0	libcrc32c	1.1.2	python	3.10.10
azure-storage-blob	12.15.0	libcups	2.3.3	python_abi	3.10
azure-storage-file-datalake	12.9.1	libcurl	7.88.1	python-dateutil	2.8.2

Library	Version	Library	Version	Library	Version
azureml-core	1.49.0	libdb	6.2.32	python-fastjsonschema	2.16.3
backcall	0.2.0	libdeflate	1.17	python-flatbuffers	23.1.21
backports	1.0	libebm	0.3.1	python-graphviz	0.20.1
backports-tempfile	1.0	libedit	3.1.20191231	python-json-logger	2.0.7
backports-weakref	1.0.post1	libev	4.33	pytorch	1.13.1
backports.functools_lru_cache	1.6.4	libevent	2.1.10	pytz	2022.7.1
bcrypt	3.2.2	libexpat	2.5.0	pyu2f	0.1.5
beautifulsoup4	4.11.2	libffi	3.4.2	pywin32-on-windows	0.1.0
bleach	6.0.0	libflac	1.4.2	pyyaml	6.0
blinker	1.6.1	libgcc-ng	12.2.0	pyzmq	25.0.2
brotli	1.0.9	libgcrypt	1.10.1	qt-main	5.15.8
brotli-bin	1.0.9	libgd	2.3.3	re2	2023.02.01
brotli-python	1.0.9	libgfortran-ng	12.2.0	readline	8.2
brotlipy	0.7.0	libgfortran5	12.2.0	regex	2022.10.31
bzip2	1.0.8	libglib	2.74.1	requests	2.28.2
c-ares	1.18.1	libgoogle-cloud	2.7.0	requests-oauthlib	1.3.1
ca-certificates	2022.12.7	libgpg-error	1.46	rfc3339-validator	0.1.4
cached_property	1.5.2	libgrpc	1.51.1	rfc3986-validator	0.1.1
cached-property	1.5.2	libhwloc	2.9.0	rsa	4.9
cachetools	5.3.0	libiconv	1.17	ruamel_yaml	0.15.80
cairo	1.16.0	liblapack	3.9.0	ruamel.yaml	0.17.21
certifi	2022.12.7	libllvm11	11.1.0	ruamel.yaml.clib	0.2.7
cffi	1.15.1	libllvm15	15.0.7	s2n	1.3.37
charset-normalizer	2.1.1	libnghttp2	1.52.0	salib	1.4.7
click	8.1.3	libnsl	2.0.0	scikit-learn	1.2.0
cloudpickle	2.2.1	libogg	1.3.4	scipy	1.10.1
colorama	0.4.6	libopenblas	0.3.21	seaborn	0.12.2
comm	0.1.3	libopus	1.3.1	seaborn-base	0.12.2
conda-package-handling	2.0.2	libpng	1.6.39	secretstorage	3.3.3

Library	Version	Library	Version	Library	Version
conda-package-streaming	0.7.0	libpq	15.2	send2trash	1.8.0
configparser	5.3.0	libprotobuf	3.21.12	setuptools	67.6.1
contextlib2	21.6.0	librsvg	2.54.4	shap	0.41.0
contourpy	1.0.7	libsndfile	1.2.0	sip	6.7.7
cryptography	40.0.1	libsodium	1.0.18	six	1.16.0
cycler	0.11.0	libsqlite	3.40.0	sleef	3.5.1
dash	2.9.2	libssh2	1.10.0	slicer	0.0.7
dash_cytoscape	0.2.0	libstdcxx-ng	12.2.0	smmap	3.0.5
dash-core-components	2.0.0	libsystemd0	253	snappy	1.1.10
dash-html-components	2.0.0	libthrift	0.18.0	sniffio	1.3.0
dash-table	5.0.0	libtiff	4.5.0	soupsieve	2.3.2.post1
databricks-cli	0.17.6	libtool	2.4.7	sqlalchemy	2.0.9
dbus	1.13.6	libudev1	253	sqlparse	0.4.3
debugpy	1.6.7	libutf8proc	2.8.0	stack_data	0.6.2
decorator	5.1.1	libuuid	2.38.1	statsmodels	0.13.5
defusedxml	0.7.1	libuv	1.44.2	synapseml-mlflow	1.0.14
dill	0.3.6	libvorbis	1.3.7	synapseml-utils	1.0.7
distlib	0.3.6	libwebp	1.2.4	tabulate	0.9.0
docker-py	6.0.0	libwebp-base	1.2.4	tbb	2021.8.0
entrypoints	0.4	libxcb	1.13	tenacity	8.2.2
et_xmlfile	1.1.0	libxgboost	1.7.1	tensorboard	2.11.2
executing	1.2.0	libxkbcommon	1.5.0	tensorboard-data-server	0.6.1
expat	2.5.0	libxml2	2.10.3	tensorboard-plugin-wit	1.8.1
fftw	3.3.10	libxslt	1.1.37	tensorflow	2.11.0
filelock	3.11.0	libzlib	1.2.13	tensorflow-base	2.11.0
flask	2.2.3	lightgbm	3.3.3	tensorflow-estimator	2.11.0
flask-compress	1.13	lime	0.2.0.1	termcolor	2.2.0
flatbuffers	22.12.06	llvm-openmp	16.0.1	terminado	0.17.1

Library	Version	Library	Version	Library	Version
flit-core	3.8.0	llvmlite	0.39.1	threadpoolctl	3.1.0
fluent-logger	0.10.0	lxml	4.9.2	tinycss2	1.2.1
font-ttf-dejavu-sans-mono	2.37	lz4-c	1.9.4	tk	8.6.12
font-ttf-inconsolata	3.000	markdown	3.4.1	toml	0.10.2
font-ttf-source-code-pro	2.038	markupsafe	2.1.2	toolz	0.12.0
font-ttf-ubuntu	0.83	matplotlib	3.6.3	tornado	6.2
fontconfig	2.14.2	matplotlib-base	3.6.3	tqdm	4.65.0
fonts-conda-ecosystem	1	matplotlib-inline	0.1.6	traitlets	5.9.0
fonts-conda-forge	1	mistune	2.0.5	treeinterpreter	0.2.2
fonttools	4.39.3	mkl	2022.2.1	typed-ast	1.4.3
freetype	2.12.1	mlflow-skinny	2.1.1	typing_extensions	4.5.0
fribidi	1.0.10	mpg123	1.31.3	typing-extensions	4.5.0
frozenlist	1.3.3	msal	1.21.0	tzdata	2023c
fsspec	2023.4.0	msal_extensions	1.0.0	unicodedata2	15.0.0
gast	0.4.0	msgpack	1.0.5	unixodbc	2.3.10
gdk-pixbuf	2.42.10	msrest	0.7.1	urllib3	1.26.14
geographiclib	1.52	msrestazure	0.6.4	virtualenv	20.19.0
geopy	2.3.0	multidict	6.0.4	wcwidth	0.2.6
gettext	0.21.1	multiprocess	0.70.14	webencodings	0.5.1
gevent	22.10.2	munkres	1.1.4	websocket-client	1.5.1
gflags	2.2.2	mypy	0.780	werkzeug	2.2.3
giflib	5.2.1	mypy-extensions	0.4.4	wheel	0.40.0
gitdb	4.0.10	mysql-common	8.0.32	widgetsnbextension	4.0.7
gitpython	3.1.31	mysql-libs	8.0.32	wrapt	1.15.0
glib	2.74.1	nbclient	0.7.3	xcb-util	0.4.0
glib-tools	2.74.1	nbconvert-core	7.3.0	xcb-util-image	0.4.0
glog	0.6.0	nbformat	5.8.0	xcb-util-keysyms	0.4.0
google-auth	2.17.2	ncurses	6.3	xcb-util-renderutil	0.3.9
google-auth-oauthlib	0.4.6	ndg-httpsclient	0.5.1	xcb-util-wm	0.4.1
google-pasta	0.2.0	nest-asyncio	1.5.6	xgboost	1.7.1

Library	Version	Library	Version	Library	Version
graphite2	1.3.13	nspr	4.35	xkeyboard-config	2.38
graphviz	2.50.0	nss	3.89	xorg-kbproto	1.0.7
greenlet	2.0.2	numba	0.56.4	xorg-libice	1.0.10
grpcio	1.51.1	numpy	1.23.5	xorg-libsm	1.2.3
gson	0.0.3	oauthlib	3.2.2	xorg-libx11	1.8.4
gst-plugins-base	1.22.0	openjpeg	2.5.0	xorg-libxau	1.0.9
gstreamer	1.22.0	openpyxl	3.1.0	xorg-libxdmcp	1.1.3
gstreamer-orc	0.4.33	openssl	3.1.0	xorg-libxext	1.3.4
gtk2	2.24.33	opt_einsum	3.3.0	xorg-libxrender	0.9.10
gts	0.7.6	orc	1.8.2	xorg-renderproto	0.11.1
h5py	3.8.0	packaging	21.3	xorg-xextproto	7.3.0
harfbuzz	6.0.0	pandas	1.5.3	xorg-xproto	7.0.31
hdf5	1.14.0	pandasql	0.7.3	xz	5.2.6
html5lib	1.1	pandocfilters	1.5.0	yaml	0.2.5
humanfriendly	10.0	pango	1.50.14	yarl	1.8.2
icu	70.1	paramiko	2.12.0	zeromq	4.3.4
idna	3.4	parquet-cpp	1.5.1	zipp	3.15.0
imageio	2.25.0	parso	0.8.3	zlib	1.2.13
importlib_metadata	5.2.0	pathos	0.3.0	zope.event	4.6
importlib_resources	5.12.0	pathspec	0.11.1	zope.interface	6.0
importlib-metadata	5.2.0	patsy	0.5.3	zstandard	0.19.0
interpret	0.3.1	pcre2	10.40	zstd	1.5.2
interpret-core	0.3.1	pexpect	4.8.0		

Default level packages for R libraries

Below you can find the table with listing all the default level packages for R and their respective versions.

Library	Version	Library	Version	Library	Version
askpass	1.1	highcharter	0.9.4	readr	2.1.3
assertthat	0.2.1	highr	0.9	readxl	1.4.1

Library	Version	Library	Version	Library	Version
backports	1.4.1	hms	1.1.2	recipes	1.0.3
base64enc	0.1-3	htmltools	0.5.3	rematch	1.0.1
bit	4.0.5	htmlwidgets	1.5.4	rematch2	2.1.2
bit64	4.0.5	httpcode	0.3.0	remotes	2.4.2
blob	1.2.3	httpuv	1.6.6	reprex	2.0.2
brew	1.0-8	httr	1.4.4	reshape2	1.4.4
brio	1.1.3	ids	1.0.1	rjson	0.2.21
broom	1.0.1	igraph	1.3.5	rlang	1.0.6
bslib	0.4.1	infer	1.0.3	rlist	0.4.6.2
cachem	1.0.6	ini	0.3.1	rmarkdown	2.18
callr	3.7.3	ipred	0.9-13	RODBC	1.3-19
caret	6.0-93	isoband	0.2.6	roxygen2	7.2.2
cellranger	1.1.0	iterators	1.0.14	rprojroot	2.0.3
cli	3.4.1	jquerylib	0.1.4	rsample	1.1.0
clipr	0.8.0	jsonlite	1.8.3	rstudioapi	0.14
clock	0.6.1	knitr	1.41	rversions	2.1.2
colorspace	2.0-3	labeling	0.4.2	rvest	1.0.3
commonmark	1.8.1	later	1.3.0	sass	0.4.4
config	0.3.1	lava	1.7.0	scales	1.2.1
conflicted	1.1.0	lazyeval	0.2.2	selectr	0.4-2
coro	1.0.3	lhs	1.1.5	sessioninfo	1.2.2
cpp11	0.4.3	lifecycle	1.0.3	shiny	1.7.3
crayon	1.5.2	lightgbm	3.3.3	slider	0.3.0
credentials	1.3.2	listenv	0.8.0	sourcetools	0.1.7
crosstalk	1.2.0	lobstr	1.1.2	sparklyr	1.7.8
crul	1.3	lubridate	1.9.0	SQUAREM	2021.1
curl	4.3.3	magrittr	2.0.3	stringi	1.7.8
data.table	1.14.6	maps	3.4.1	stringr	1.4.1
DBI	1.1.3	memoise	2.0.1	sys	3.4.1
dbplyr	2.2.1	mime	0.12	systemfonts	1.0.4

Library	Version	Library	Version	Library	Version
desc	1.4.2	miniUI	0.1.1.1	testthat	3.1.5
devtools	2.4.5	modeldata	1.0.1	textshaping	0.3.6
dials	1.1.0	modelenv	0.1.0	tibble	3.1.8
DiceDesign	1.9	ModelMetrics	1.2.2.2	tidymodels	1.0.0
diffobj	0.3.5	modelr	0.1.10	tidyr	1.2.1
digest	0.6.30	munsell	0.5.0	tidyselect	1.2.0
downlit	0.4.2	numDeriv	2016.8-1.1	tidyverse	1.3.2
dplyr	1.0.10	openssl	2.0.4	timechange	0.1.1
dtplyr	1.2.2	parallelly	1.32.1	timeDate	4021.106
e1071	1.7-12	parsnip	1.0.3	tinytex	0.42
ellipsis	0.3.2	patchwork	1.1.2	torch	0.9.0
evaluate	0.18	pillar	1.8.1	triebeard	0.3.0
fansi	1.0.3	pkgbuild	1.4.0	TTR	0.24.3
farver	2.1.1	pkgconfig	2.0.3	tune	1.0.1
fastmap	1.1.0	pkgdown	2.0.6	tzdb	0.3.0
fontawesome	0.4.0	pkgload	1.3.2	urlchecker	1.0.1
forcats	0.5.2	plotly	4.10.1	urltools	1.7.3
foreach	1.5.2	plyr	1.8.8	usethis	2.1.6
forge	0.2.0	praise	1.0.0	utf8	1.2.2
fs	1.5.2	prettyunits	1.1.1	uuid	1.1-0
furrr	0.3.1	pROC	1.18.0	vctrs	0.5.1
future	1.29.0	processx	3.8.0	viridisLite	0.4.1
future.apply	1.10.0	prodlim	2019.11.13	vroom	1.6.0
gargle	1.2.1	profvis	0.3.7	waldo	0.4.0
generics	0.1.3	progress	1.2.2	warp	0.2.0
gert	1.9.1	progressr	0.11.0	whisker	0.4
ggplot2	3.4.0	promises	1.2.0.1	withr	2.5.0
gh	1.3.1	proxy	0.4-27	workflows	1.1.2
gistr	0.9.0	pryr	0.1.5	workflowsets	1.0.0
gitcreds	0.1.2	ps	1.7.2	xfun	0.35

Library	Version	Library	Version	Library	Version
globals	0.16.2	purrr	0.3.5	xgboost	1.6.0.1
glue	1.6.2	quantmod	0.4.20	XML	3.99-0.12
googledrive	2.0.0	r2d3	0.2.6	xml2	1.3.3
googlesheets4	1.0.1	R6	2.5.1	xopen	1.0.0
gower	1.0.0	ragg	1.2.4	xtable	1.8-4
GPfit	1.0-8	rappdirs	0.3.3	xts	0.12.2
gtable	0.3.1	rbokeh	0.5.2	yaml	2.3.6
hardhat	1.2.0	rcmdcheck	1.4.0	yardstick	1.1.0
haven	2.5.1	RColorBrewer	1.1-3	zip	2.2.2
hexbin	1.28.2	Rcpp	1.0.9	zoo	1.8-11

Migration between different Apache Spark Versions

Migrating your workloads to Fabric Runtime 1.1 (Apache Spark 3.3) from an older version of Apache Spark involves a series of steps to ensure a smooth migration. This guide outlines the necessary steps to help you migrate efficiently and effectively.

1. Review Fabric Runtime 1.1 release notes, including checking the components and default-level packages included into the runtime, to understand the new features, improvements.

2. Check compatibility of your current setup and all related libraries, including dependencies and integrations. Review the migration guides to identify potential breaking changes:

   - Review Spark Core migration guide
   - Review SQL, Datasets and DataFrame migration guide
   - If your solution is Apache Spark Structure Streaming related, review Structured Streaming migration guide
   - If you use PySpark, review Pyspark migration guide
   - If you migrate code from Koalas to PySpark, review Koalas to pandas API on Spark migration guide

3. Move your workloads to Fabric and ensure that you have backups of your data and configuration files in case you need to revert to the previous version.

4. Update any dependencies that may be impacted by the new version of Apache Spark or other Fabric Runtime 1.1 related components. This could include third-party libraries or connectors. Make sure to test the updated dependencies in a staging environment before deploying to production

5. Update Apache Spark Configuration on your workload. This could include updating configuration settings, adjusting memory allocations, and modifying any deprecated

configurations.

6. Modify your Apache Spark applications (notebooks and Apache Spark Jobs Definitions) to use the new APIs and features introduced in Fabric Runtime 1.1 and Apache Spark 3.3. This may involve updating your code to accommodate any deprecated or removed APIs, and refactoring your applications to take advantage of performance improvements and new functionalities.
7. Thoroughly test your updated applications in a staging environment to ensure compatibility and stability with Apache Spark 3.3. Perform performance testing, functional testing, and regression testing to identify and resolve any issues that may arise during the migration process.
8. After validating your applications in a staging environment, deploy the updated applications to your production environment. Monitor the performance and stability of your applications after the migration to identify any issues that need to be addressed.
9. Update your internal documentation and training materials to reflect the changes introduced in Fabric Runtime 1.1. Ensure that your team members are familiar with the new features and improvements to maximize the benefits of the migration.

How to create custom Spark pools in Microsoft Fabric

Article • 05/23/2023

In this document, we'll explain how to create custom Apache Spark pools in Microsoft Fabric for your analytics workloads. Apache Spark pools enable users to create tailored compute environments based on their specific requirements, ensuring optimal performance and resource utilization.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

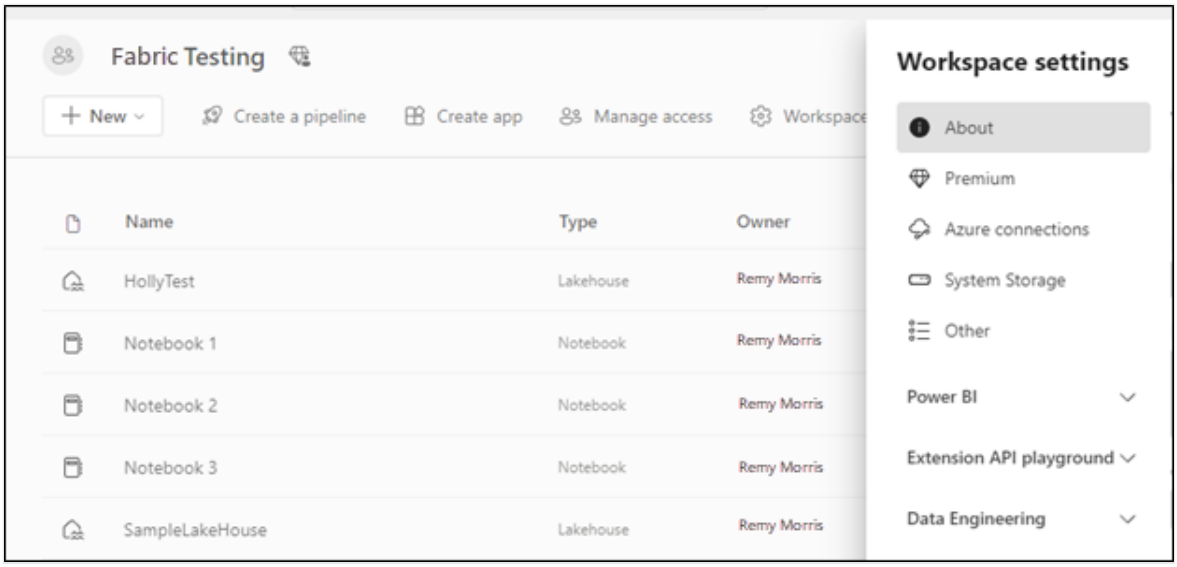
Users specify the minimum and maximum nodes for autoscaling. Based on which, the system dynamically acquires and retires nodes as the job's compute requirements change. It results in efficient scaling and improved performance. Furthermore, the dynamic allocation of executors in Spark pools alleviates the need for manual executor configuration. Instead, the system adjusts the number of executors depending on the data volume and job-level compute needs. This way, it enables you to focus on your workloads without worrying about performance optimization and resource management.

> **Note**
>
> To create a custom spark pool, you must have admin access to the workspace. The capacity admin should have enabled the **Customized workspace pools** option in the **Spark Compute** section of **Capacity Admin settings**. To learn more, see **Spark Compute Settings for Fabric Capacities**.

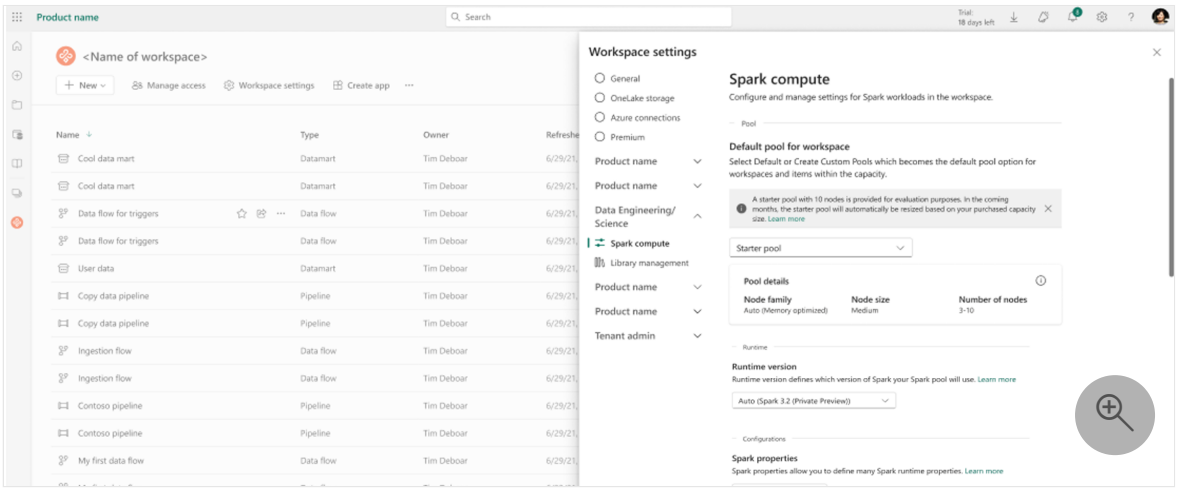
Create custom Spark pools

To create or manage the Spark Pool associated with your workspace:

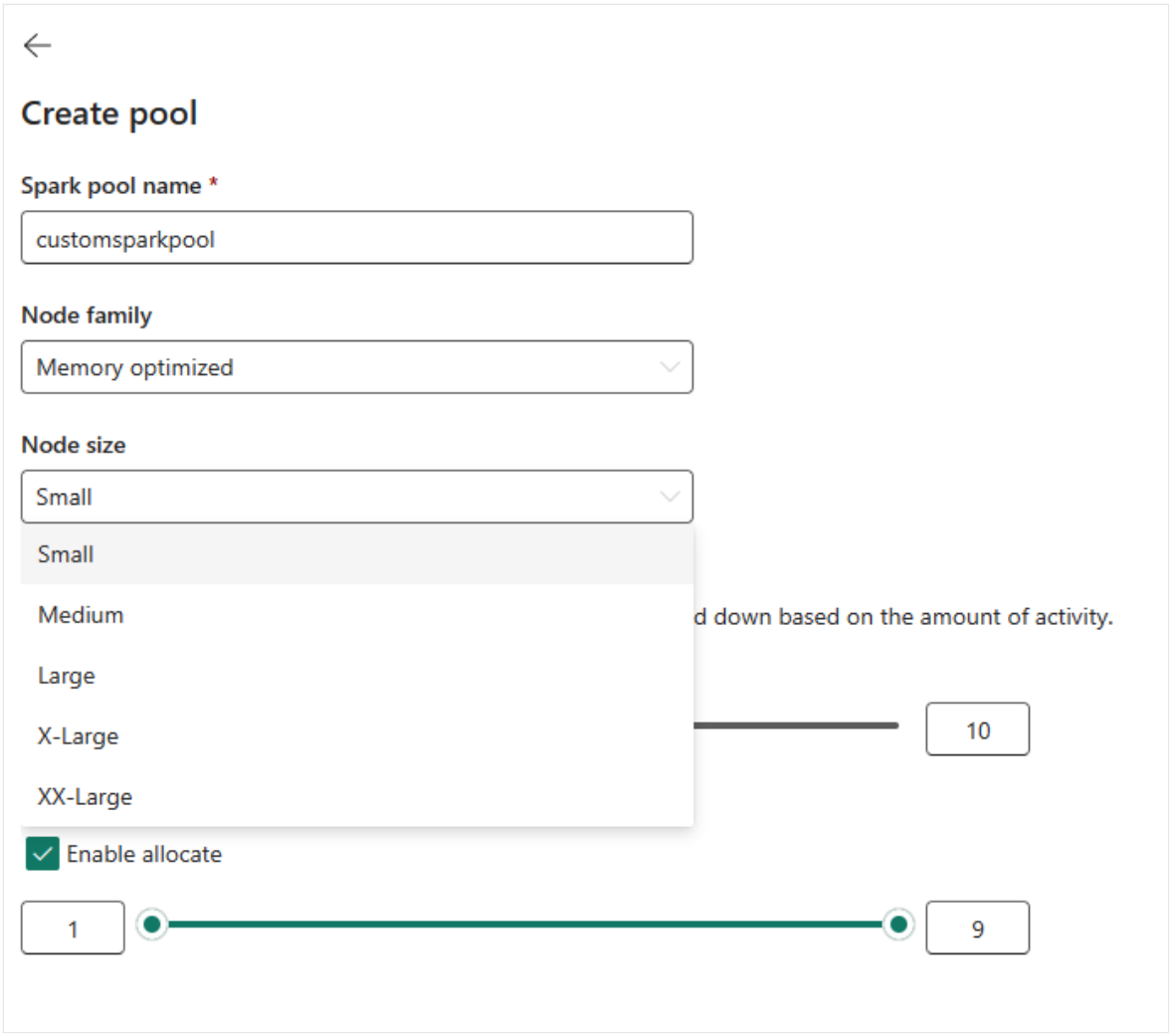
1. Go to your workspace and choose the **Workspace settings**:



2. Then, select the **Data Engineering/Science** option to expand the menu. Navigate to the **Spark Compute** option from the left-hand menu:

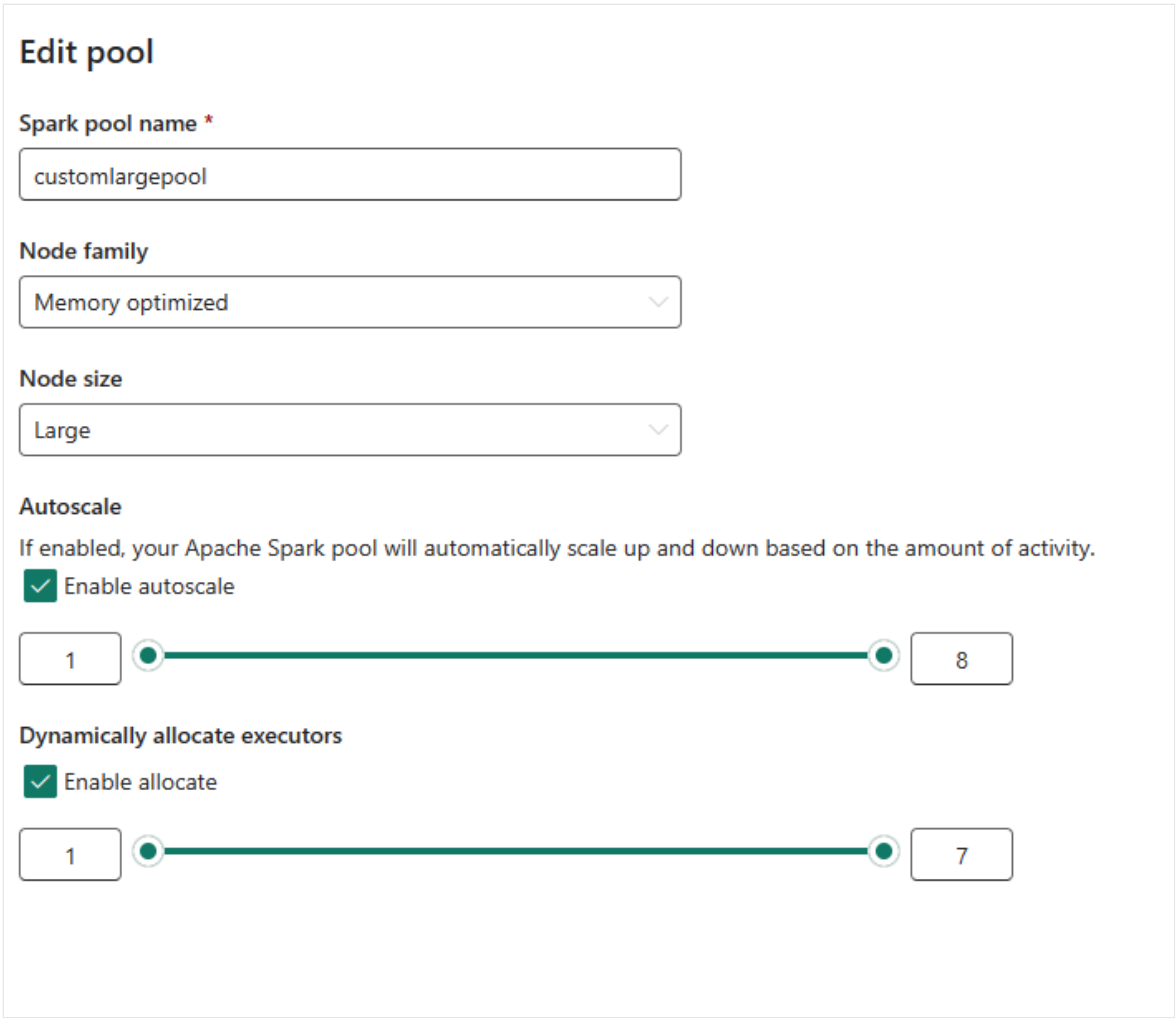


3. Select the **New Pool** option. From the **Create Pool** menu, name your Spark pool. Select the **Node family**, and **Node size** from the available sizes Small, Medium, Large, X-Large and XX-Large based on compute requirements for your workloads.



4. You can also set the minimum node configuration for your custom pools to 1. Because the Fabric Spark provides restorable availability for clusters with single node, you do not have to worry about job failures, loss of session during failures, or over paying on compute for smaller spark jobs.

5. You can also enable or disable autoscaling for your custom Spark pools. When autoscaling is enabled, the pool will dynamically acquire new nodes up to the maximum node limit specified by the user, and then retire them after job execution. This feature ensures better performance by adjusting resources based on the job requirements. You are allowed to size the nodes, which fit within the capacity units purchased as part of the Fabric capacity SKU.



6. You can also choose to enable dynamic executor allocation for your Spark pool, which automatically determines the optimal number of executors within the user-specified maximum bound. This feature adjusts the number of executors based on data volume, resulting in improved performance and resource utilization.

7. These custom pools have a default auto-pause duration of 2 minutes. Once the auto-pause duration is reached, the session expires and the clusters are unallocated. You are charged based on the number of nodes and the duration for which the custom spark pools are used.

Next steps

- Learn more from the Apache Spark public documentation.
- Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace

What is autotune for Apache Spark configurations in Fabric and how to enable and disable it?

Article • 05/23/2023

Autotune automatically tunes Apache Spark configurations to minimize workload execution time and optimizes workloads. It empowers you to achieve more with less. This feature reduces execution time and surpasses the gains accomplished by manually tuned workloads by experts, which necessitate considerable effort and experimentation.

It leverages historical data execution from your workloads to iteratively learn the optimal configurations for a given workload and its execution time.

> ⓘ **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Query tuning

Currently, autotune configures three query-level of Apache Spark configurations:

- `spark.sql.shuffle.partitions` - configures the number of partitions to use when shuffling data for joins or aggregations. Default is 200.
- `spark.sql.autoBroadcastJoinThreshold` - configures the maximum size in bytes for a table that will be broadcasted to all worker nodes when performing a join. Default is 10 MB.
- `spark.sql.files.maxPartitionBytes` - the maximum number of bytes to pack into a single partition when reading files. Works for Parquet, JSON and ORC file-based sources. Default is 128 MB.

Since there's no historical data available during the first run of autotune, configurations will be set based on a baseline model. This model relies on heuristics related to the content and structure of the workload itself. However, as the same query or workload is run repeatedly, we'll observe increasingly significant improvements from autotune. As the results of previous runs are used to fine-tune the model and tailor it to a specific workspace or workload.

> **Note**
>
> As the algorithm explores various configurations, you may notice minor differences in results. This is expected, as autotune operates iteratively and improves with each repetition of the same query.

Configuration tuning algorithm overview

For the first run of the query, upon submission, a machine learning (ML) model initially trained using standard open-source benchmark queries (e.g., TPC-DS) will guide the search around the neighbors of the current setting (starting from the default). Among the neighbor candidates, the ML model selects the best configuration with the shortest predicted execution time. In this run, the "centroid" is the default config, around which the autotune generates new candidates.

Based on the performance of the second run per suggested configuration, we retrain the ML model by adding the new observation from this query, and update the centroid by comparing the performance of the last two runs. If the previous run is better, the centroid will be updated in the inverse direction of the previous update (similar to the momentum approach in DNN training); if the new run is better, the latest configuration setting becomes the new centroid. Iteratively, the algorithm will gradually search in the direction with better performance.

Enable or disable autotune

Autotune is disabled by default and it's controlled by Apache Spark Configuration Settings. Easily enable Autotune within a session by running the following code in your notebook or adding it in your spark job definition code:

Spark SQL

```
SQL

%%sql
SET spark.ms.autotune.queryTuning.enabled=TRUE
```

To verify and confirm its activation, use the following commands:

Spark SQL

```
SQL

%%sql
GET spark.ms.autotune.queryTuning.enabled
```

To disable Autotune, execute the following commands:

Spark SQL

```
SQL

%%sql
SET spark.ms.autotune.queryTuning.enabled=FALSE
```

Transparency note

Microsoft follows Responsible AI Standard and this transparency note aims to provide clear documentation defining the intended uses of Autotune and the evidence that the feature is fit for purpose before the service becomes externally available. We understand the importance of transparency and ensuring that our customers have the necessary information to make informed decisions when using our services.

Intended uses of the Autotune

The primary goal of Autotune is to optimize the performance of Apache Spark workloads by automating the process of Apache Spark configuration tuning. The system is designed to be used by data engineers, data scientists, and other professionals who are involved in the development and deployment of Apache Spark workloads. The intended uses of the Autotune include:

- Automatic tuning of Apache Spark configurations to minimize workload execution time to accelerate development process
- Reducing the manual effort required for Apache Spark configuration tuning
- Leveraging historical data execution from workloads to iteratively learn optimal configurations

Evidence that the Autotune is fit for purpose

To ensure that Autotune meets the desired performance standards and is fit for its intended use, we have conducted rigorous testing and validation. The evidence includes:

1. Thorough internal testing and validation using various Apache Spark workloads and datasets to confirm the effectiveness of the autotuning algorithms
2. Comparisons with alternative Apache Spark configuration optimization techniques, demonstrating the performance improvements and efficiency gains achieved by Autotune
3. Customer case studies and testimonials showcasing successful applications of Autotune in real-world projects
4. Compliance with industry-standard security and privacy requirements, ensuring the protection of customer data and intellectual property

We want to assure that we prioritize data privacy and security. Your data will only be used to train the model that serves your specific workload. We take stringent measures to ensure that no sensitive information is used in our storage or training processes.

Concurrency limits and queueing in Microsoft Fabric Spark

Article • 05/23/2023

Applies to: ✅ Data Engineering and Data Science in Microsoft Fabric

Microsoft Fabric allows allocation of compute units through capacity, which is a dedicated set of resources that is available at a given time to be used. Capacity defines the ability of a resource to perform an activity or to produce output. Different items consume different capacity at a certain time. Microsoft Fabric offers capacity through the Fabric SKUs and trials. For more information, see What is capacity?

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

When users create a Microsoft Fabric capacity on Azure, they get to choose a capacity size based on their analytics workload size. In Spark, users get two spark VCores for every capacity unit they get reserved as part of their SKU.

One Capacity Unit = Two Spark VCores

Once the capacity is purchased, admins can create workspaces within the capacity in Microsoft Fabric. The Spark VCores associated with the capacity is shared among all the Spark-based items like notebooks, spark job definitions, and the lakehouse created in these workspaces.

Concurrency throttling and queueing

The following section lists various numerical limits for Spark workloads based on Microsoft Fabric capacity SKUs:

Capacity SKU	Equivalent Power BI SKU	Capacity Units	Equivalent Spark VCores	Max Concurrent Jobs	Queue Limit
F2	-	2	4	1	4
F4	-	4	8	1	4

Capacity SKU	Equivalent Power BI SKU	Capacity Units	Equivalent Spark VCores	Max Concurrent Jobs	Queue Limit
F8	-	8	16	2	8
F16	-	16	32	5	20
F32	-	32	64	10	40
F64	P1	64	128	20	80
Fabric Trial	P1	64	128	5	-
F128	P2	128	256	40	160
F256	P3	256	512	80	320
F512	P4	512	1024	160	640

The queueing mechanism is a simple FIFO-based queue, which checks for available job slots and automatically retries the jobs once the capacity has become available. As there are different items like notebooks, spark job definition, and lakehouse which users could use in any workspace. As the usage varies across different enterprise teams, users could run into starvation scenarios where there is dependency on only type of item, such as a spark job definition. This could result in users sharing the capacity from running a notebook-based job or any lakehouse based operation like load to table.

To avoid these blocking scenarios, Microsoft Fabric applies a **Dynamic reserve based throttling** for jobs from these items. Notebook and lakehouse based jobs being more interactive and real-time are classified as **interactive**. Whereas Spark job definition is classified as **batch**. As part of this dynamic reserve, minimum and maximum reserve bounds are maintained for these job types. The reserves are mainly to address use cases where an enterprise team could experience peak usage scenarios having their entire capacity consumed through batch jobs. During those peak hours, users are blocked from using interactive items like notebooks or lakehouse. With this approach, every capacity gets a minimum reserve of 30% of the total jobs allocated for interactive jobs (5% for lakehouse and 25% for notebooks) and a minimum reserve of 10% for batch jobs.

Job Type	Item	Min %	Max %
Batch	Spark Job Definition	10	70
Interactive	Interactive Min and Max	30	90

Job Type	Item	Min %	Max %
	Notebook	25	85
	Lakehouse	5	65

When they exceed these reserves and when the capacity is at its maximum utilization, interactive jobs like notebooks and lakehouse are throttled with the message *HTTP Response code 430: Unable to submit this request because all the available capacity is currently being used. Cancel a currently running job, increase your available capacity, or try again later*.

With queueing enabled, batch jobs like Spark Job Definitions get added to the queue and are automatically retried when the capacity is freed up.

> **Note**
>
> The jobs have a queue expiration period of 24 hours, after which they are cancelled and users would have to resubmit them for job execution.

Next steps

- Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace
- Learn about the Spark Compute for Fabric Data Engineering/Science experiences

What is an Apache Spark job definition?

Article • 05/23/2023

An Apache Spark Job Definition is a Microsoft Fabric code item that allows you to submit batch/streaming job to Spark cluster. By uploading the binary files from compilation output of different languages, .jar from Java for example, you can apply different transformation logic to the data hosted on lakehouse. Besides the binary file, you can further customize the behavior of the job by uploading additional libraries and command line arguments.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

To run a Spark job definition, you must have at least one lakehouse associated with it. This default lakehouse context serves as the default file system for Spark runtime. For any Spark code using relative path to read/write data, the data is served from the default lakehouse.

> **Tip**
>
> To run the Spark job definition item, main definition file and default lakehouse context are required. If you don't have a lakehouse, you can create one by following the steps in **Create a lakehouse**.

> **Important**
>
> The Spark job definition item is currently in PREVIEW.

Next steps

In this overview, you get a basic understanding of a Spark job definition. Advance to the next article to learn how to create and get started with your own Spark job definition:

- To get started with Microsoft Fabric, see Creating an Apache Spark job definition.

How to create an Apache Spark job definition in Fabric

Article • 05/23/2023

In this tutorial, learn how to create a Spark job definition in Microsoft Fabric.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Prerequisites

To get started, you need the following prerequisites:

- A Microsoft Fabric tenant account with an active subscription. Create an account for free.

> **Tip**
>
> To run the Spark job definition item, main definition file and default lakehouse context are required. If you don't have a lakehouse, you can create one by following the steps in **Create a lakehouse**.

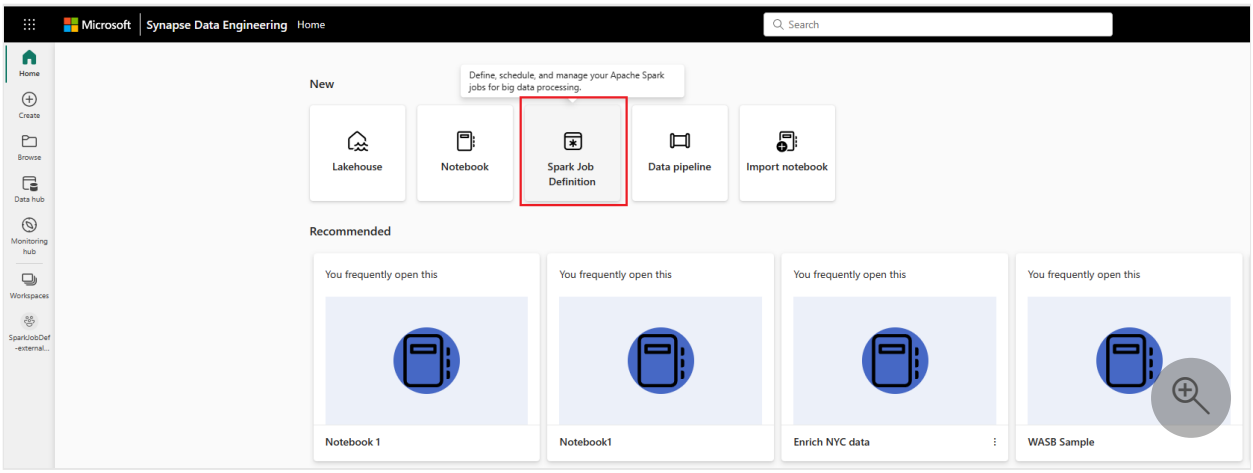
Create a Spark job definition

The Spark job definition creation process is quick and simple and there are several ways to get started.

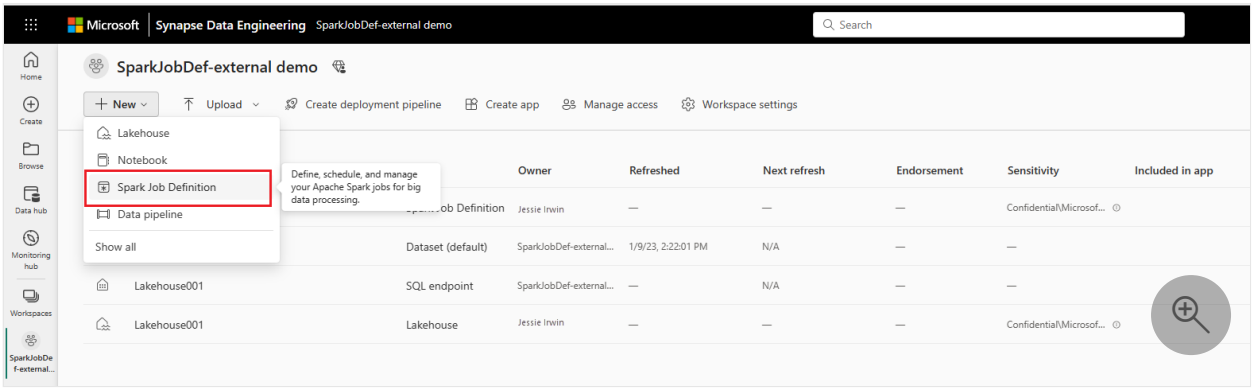
Options to create a Spark job definition

There are a few ways you can get started with the creation process:

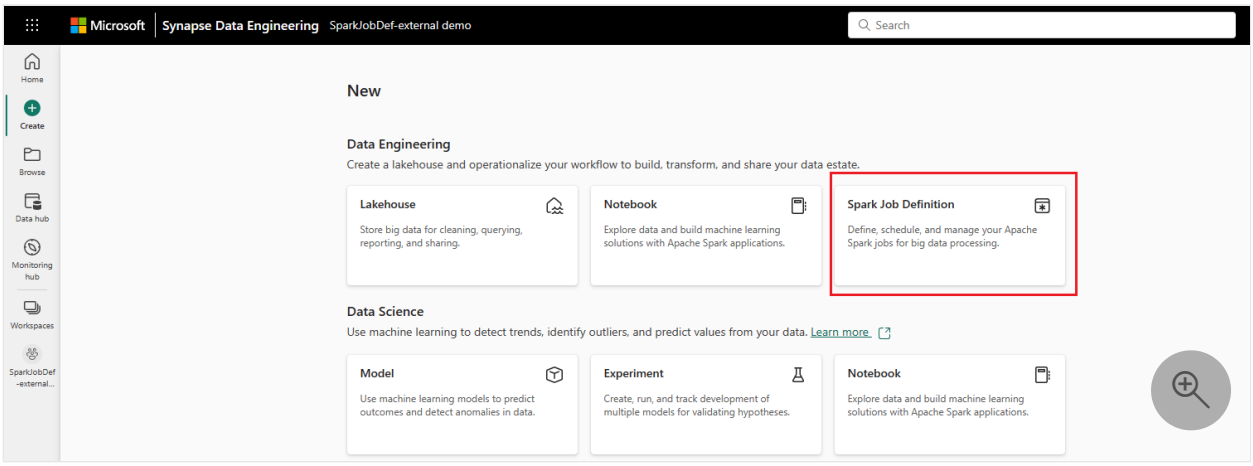
- **Data engineering homepage**: You can easily create a Spark job definition through the **Spark job definition** card under the **New** section in the homepage.



- **Workspace view**: You can also create a Spark job definition through the **Workspace** view when you are on the **Data Engineering** experience by using the **New** dropdown.



- **Create Hub**: Another entry point to create a Spark job definition is in the **Create Hub** page under **Data Engineering**.



A name would be required to create a Spark job definition. The name must be unique within the current workspace. The newly created Spark Job definition will be created under the current workspace you are in.

Create a Spark job definition for PySpark (Python)

To create a Spark job definition for PySpark, follow these steps:

1. Create a new Spark job definition.

2. Select **PySpark (Python)** from the **Language** dropdown.

3. Upload the main definition file as *.py* file. The main definition file is the file that contains the application logic of this *job.Main* definition file is mandatory to run a Spark job. For each Spark Job Definition, you can only upload one main definition file.

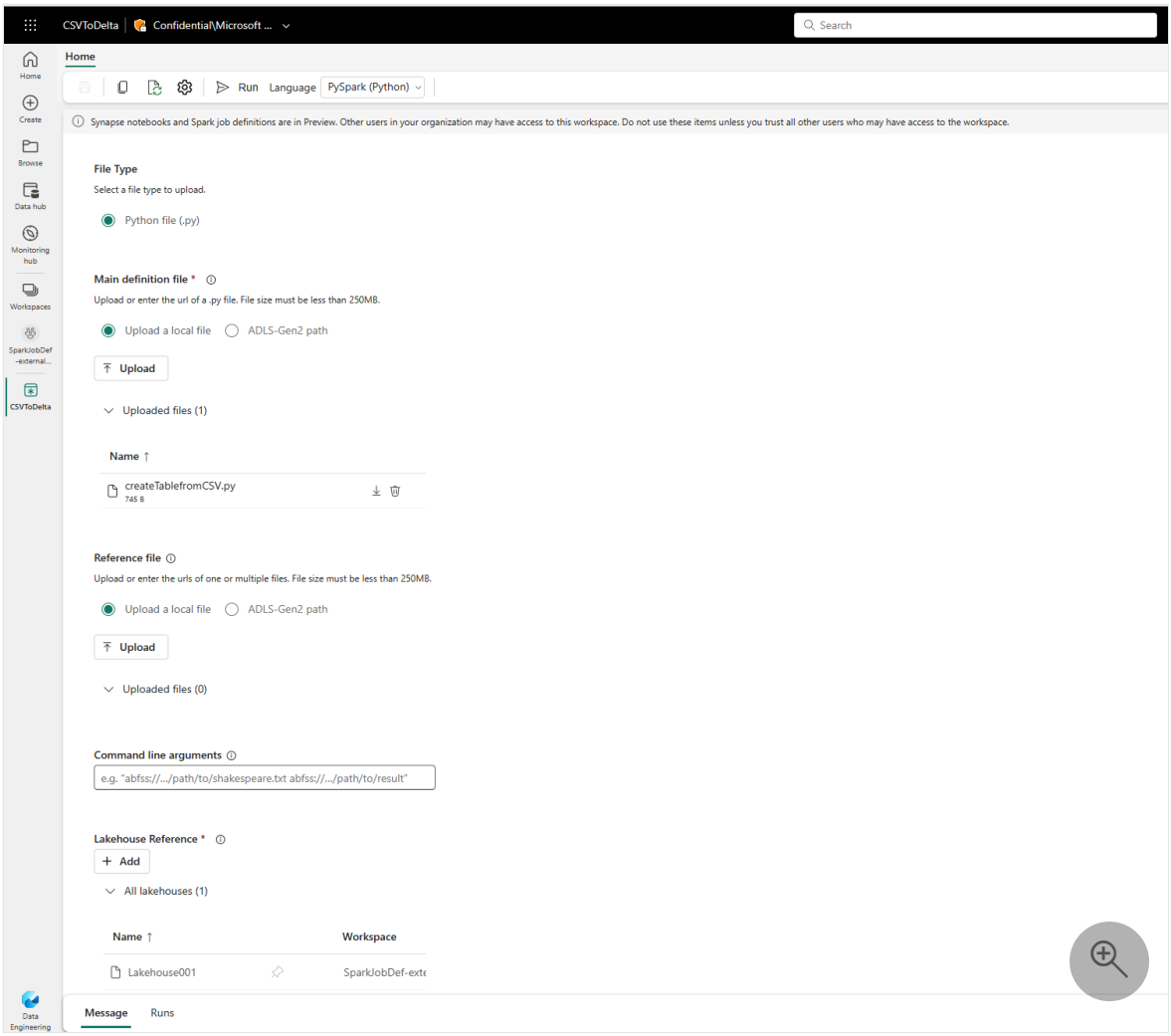
   Beside uploading from local desktop, you can also upload from existing Azure Data Lake Storage Gen2 by providing the full abfss path of the file. For example, abfss://your-storage-account-name.dfs.core.windows.net/your-file- path.

4. Upload Reference files as *.py* file. the Reference files are the python modules that are imported by the main definition file. Similar as uploading main definition file, you can also upload from existing Azure Data Lake Storage Gen2 by providing the full abfss path of the file. Multiple reference files are supported.

> **Tip**
>
> If ADLS-gen2 path is used, to make sure the file is accessible, The user account which is used to run the job should be assigned with proper permission to the storage account. There are two suggested way to do this:
>
> - Assign the user account as Contributor role to the storage account.
> - Grant Read and Execution permission to the user account on the file via Azure Data Lake Storage Gen2 Access Control List (ACL)
>
> For manually run, the account of current login user would be used to run the job

5. Provide command line arguments to the job if needed. please use space as splitter to separate the arguments.

6. Add the lakehouse reference to the job. You must have at least one lakehouse reference added to the job. This lakehouse is the default lakehouse context for the job. Multiple lakehouse references are supported. For the non-default Lakehouse, you can find its name and full OneLake URL in the Spark Settings page.



In this example, we've done the following:

- Created a Spark job definition named **CSVToDelta** for PySpark
- Uploaded the *createTablefromCSV.py* file as the main definition file
- Added the lakehouse references *LH001* and *LH002* to the job
- Made *LH001* the default lakehouse context

Create a Spark job definition for Scala/Java

To create a Spark job definition for Scala/Java, follow these steps:

1. Select **Spark(Scala/Java)** from the **Language** dropdown.

2. Upload the main definition file as .jar file. The main definition file is the file that contains the application logic of this job. A main definition file is mandatory to run a Spark Job. Provide the Main class name.

3. Upload Reference files as .jar file. the Reference files are the files that are referenced/imported by the main definition file.

4. Provides command line arguments to the job if needed.

5. Add the lakehouse reference to the job. You must have at least one lakehouse reference added to the job. This lakehouse is the default lakehouse context for the job.

Create a Spark job definition for R

To create a Spark job definition for SparkR(R), follow these steps:

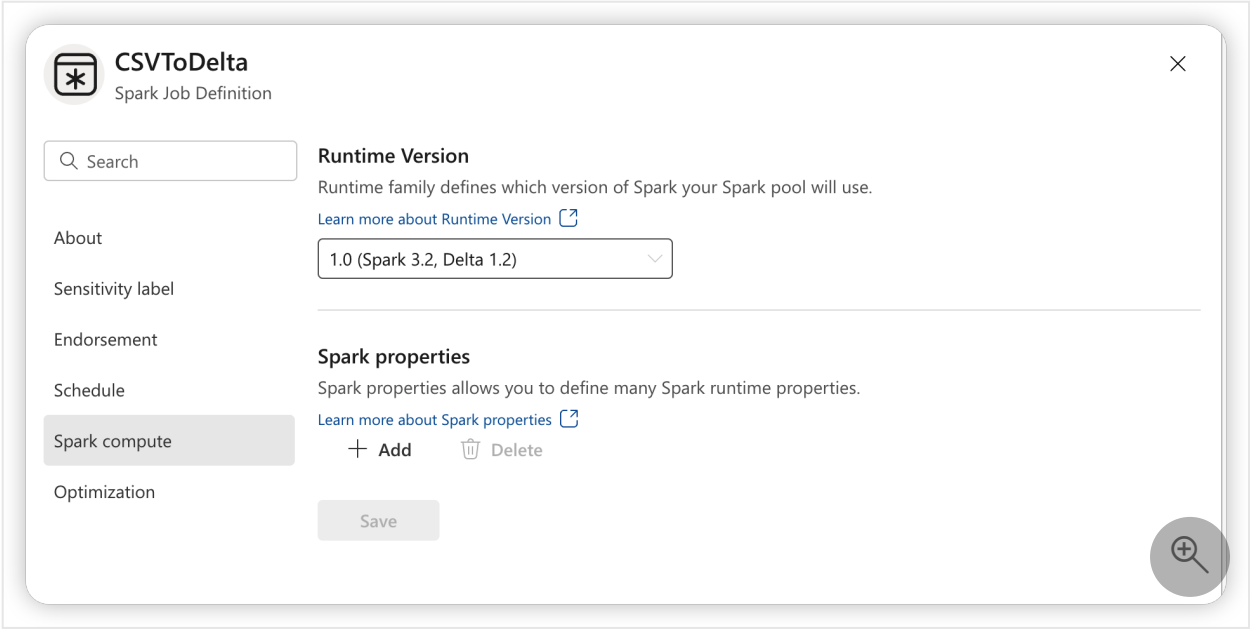
1. Select **SparkR(R)** from the **Language** dropdown.
2. Upload the main definition file as .R file. The main definition file is the file that contains the application logic of this job. A main definition file is mandatory to run a Spark Job.
3. Upload Reference files as .R file. the Reference files are the files that are referenced/imported by the main definition file.
4. Provides command line arguments to the job if needed.
5. Add the lakehouse reference to the job. You must have at least one lakehouse reference added to the job. This lakehouse is the default lakehouse context for the job.

> ⓘ **Note**
>
> The Spark job definition will be created under the current workspace you are in.

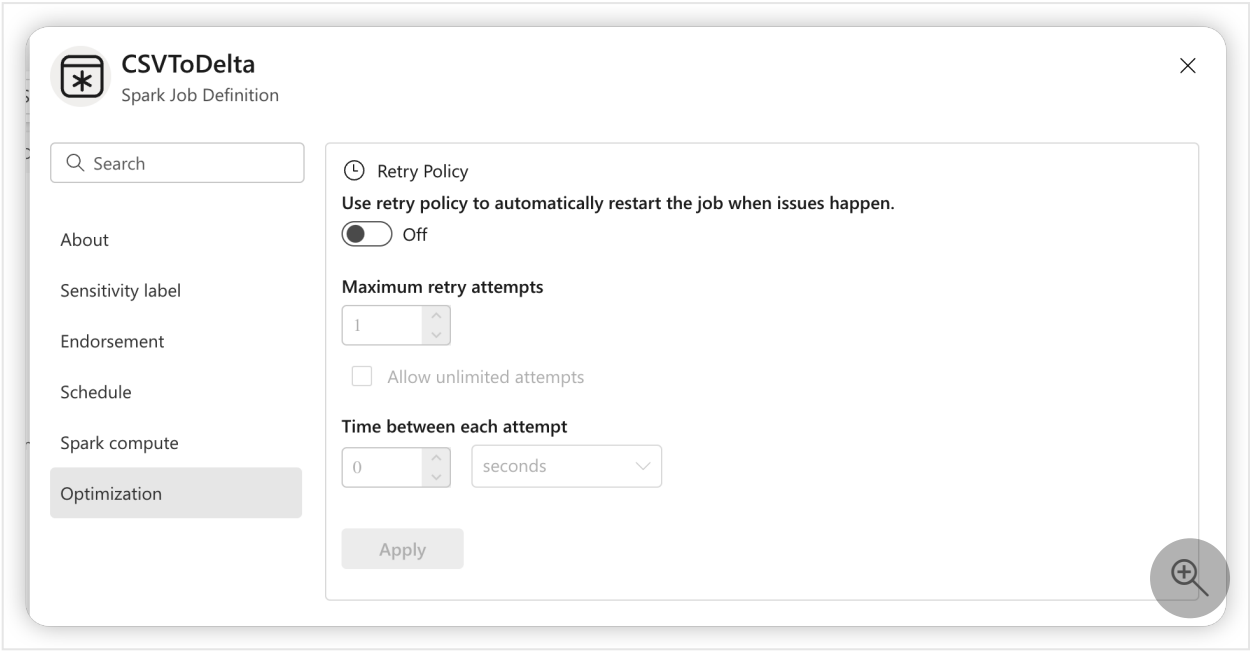
Options to customize Spark job definition

There are a few options to further customize the execution of Spark job definition

- **Spark Compute**: Within the **Spark Compute** tab, you can see the Runtime Version which is the version of Spark that will be used to run the job. You can also see the Spark configuration settings that will be used to run the job. You can customize the Spark configuration settings by clicking on the **Add** button.



- **Optimization**: Within the **Optimization** tab, you can enable and set up the Retry Policy for the job. When enabled, the job will be retried if it fails. You can also set the maximum number of retries and the interval between retries. For each attempt of retry, the job will be restarted, please make sure the job is idempotent.



Next steps

- Run an Apache Spark job definition

Schedule and run an Apache Spark job definition

Article • 05/23/2023

In this tutorial, learn how to run a Microsoft Fabric Spark job definition item and monitor the job.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Prerequisites

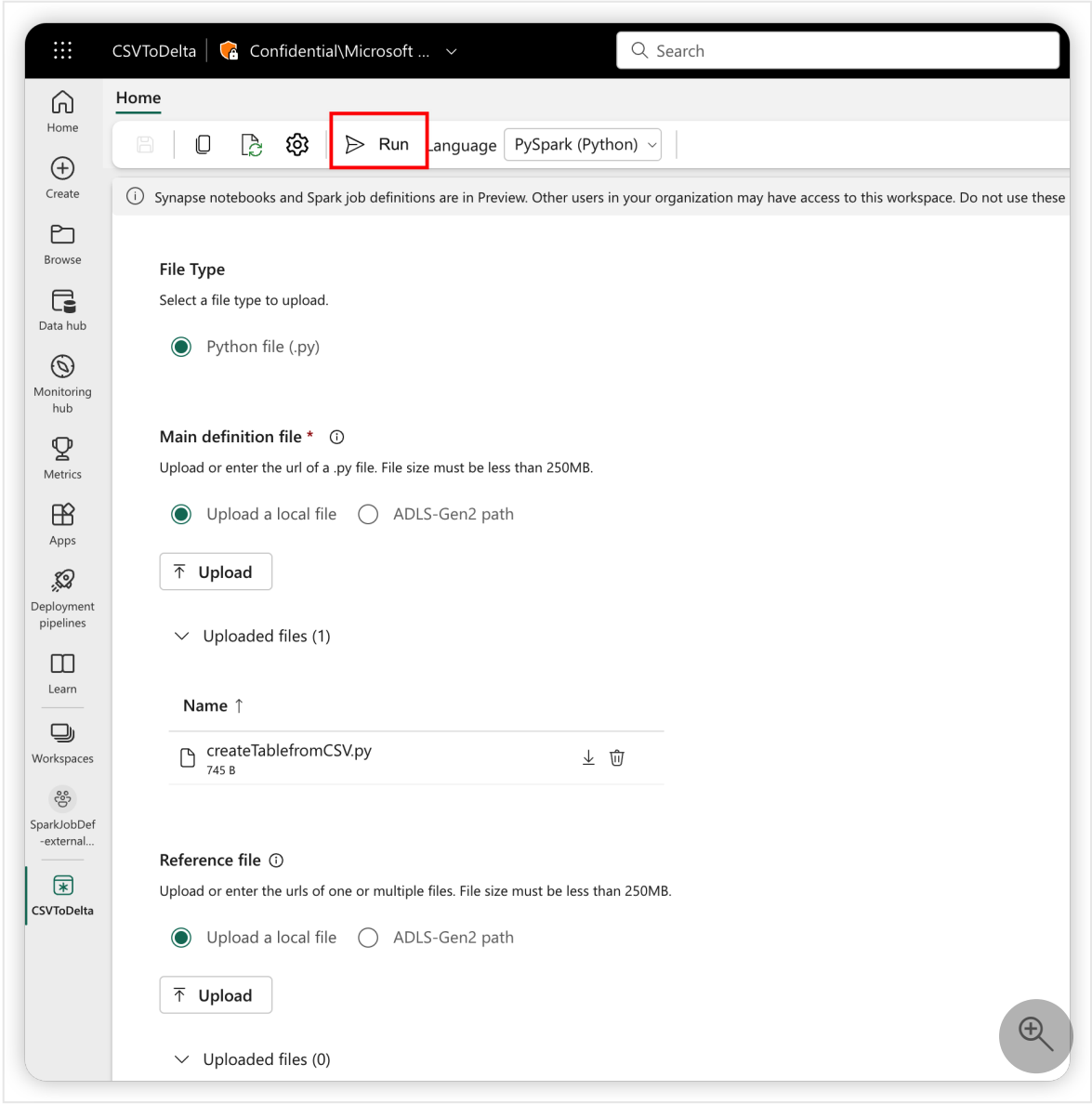
To get started, you must have the following prerequisites:

- A Microsoft Fabric tenant account with an active subscription. Create an account for free.
- Understand the Spark job definition: What is an Apache Spark job definition?
- Create a Spark job definition: How to create an Apache Spark job definition.

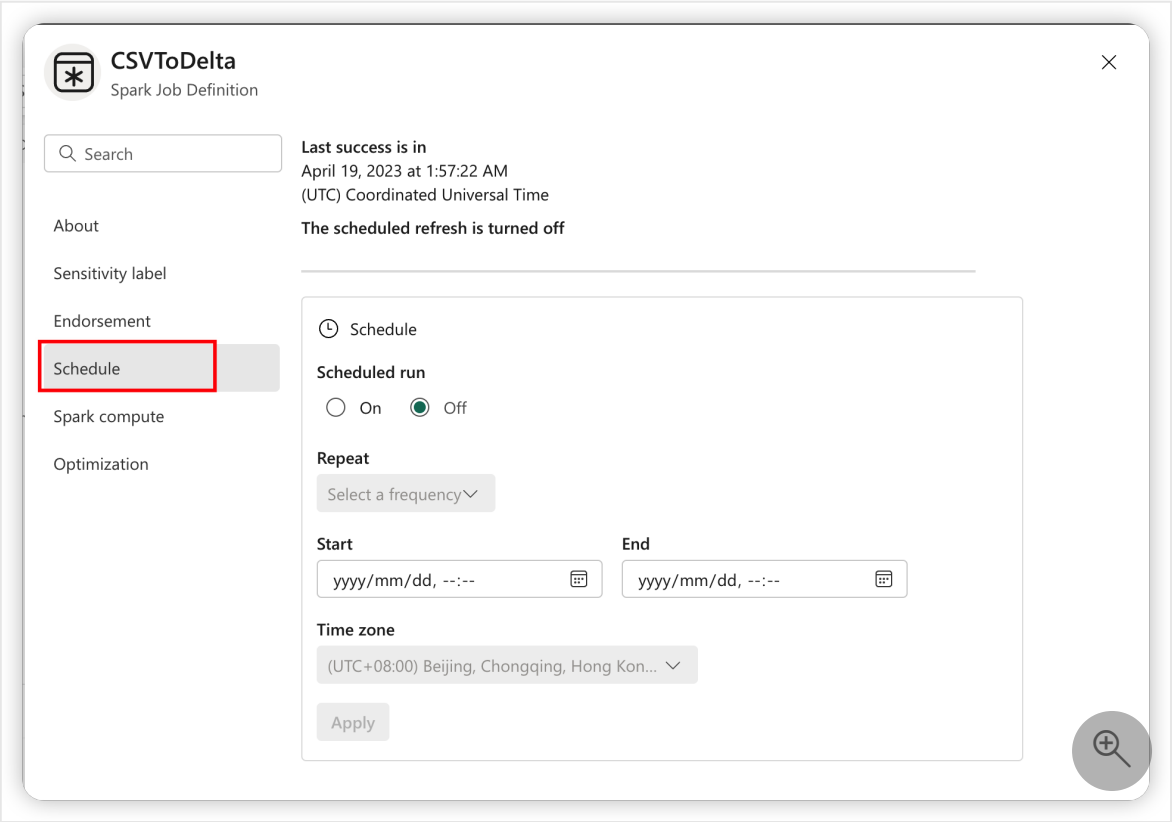
How to run a Spark job definition

There are two ways a user could run a Spark job definition:

- Run a Spark job definition item manually by clicking the **Run** button on the Spark job definition item.



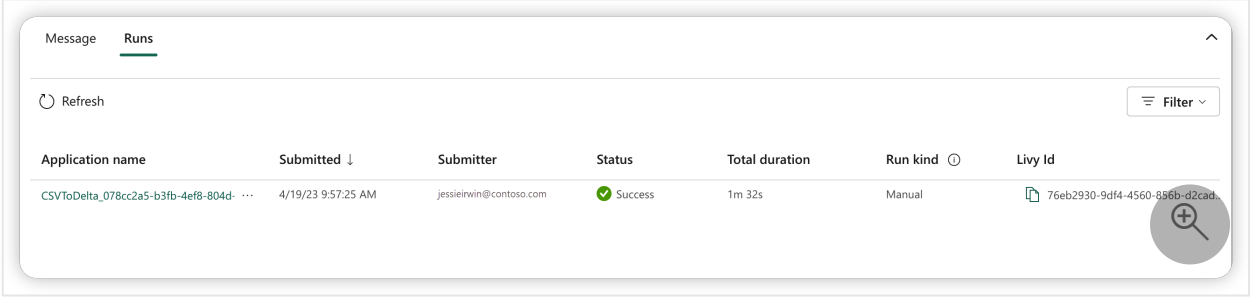
- Schedule a Spark job definition item by setting up the schedule plan under the **Settings** tab. Select **Settings** on the toolbar, then select the **Schedule** tab.



> **Important**
>
> To run a Spark job definition, it must have the main definition file and the default lakehouse context.

> **Tip**
>
> For the run triggered by the "Run" button, the account of current login user will be used to submit the job For the run triggered by the schedule plan, the account of the user who setup the schedule plan will be used to submit the job

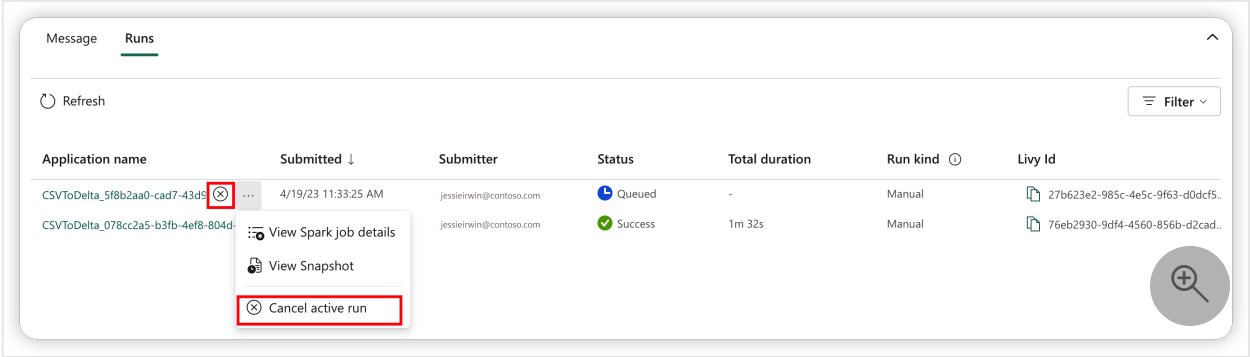
Once you've submitted the run, after three to five seconds, a new row appears under the **Runs** tab. The row shows details about your new run. The **Status** column shows the near real-time status of the job and the **Run Kind** column shows if the job is manual or scheduled.



For the detail of how to monitor the job, see Monitor a Spark job.

How to cancel a running job

Once the job is submitted, you can cancel the job by clicking the **Cancel** button on the Spark job definition item from the job list



Next steps

- Advanced capabilities: Microsoft Apache Spark utilities

Introduction of Fabric MSSparkUtils

Article • 05/23/2023

Microsoft Spark Utilities (MSSparkUtils) is a built-in package to help you easily perform common tasks. You can useMSSparkUtils to work with file systems, to get environment variables, to chain notebooks together, and to work with secrets. MSSparkUtils are available in PySpark (Python) Scala, SparkR notebooks and Microsoft Fabric pipelines.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

File system utilities

mssparkutils.fs provides utilities for working with various file systems, including Azure Data Lake Storage Gen2 (ADLS Gen2) and Azure Blob Storage. Make sure you configure access to Azure Data Lake Storage Gen2 and Azure Blob Storage appropriately.

Run the following commands for an overview of the available methods:

Python

```
from notebookutils import mssparkutils
mssparkutils.fs.help()
```

Output

Console

```
mssparkutils.fs provides utilities for working with various FileSystems.

Below is overview about the available methods:

cp(from: String, to: String, recurse: Boolean = false): Boolean -> Copies a file or directory, possibly across FileSystems
mv(from: String, to: String, recurse: Boolean = false): Boolean -> Moves a file or directory, possibly across FileSystems
ls(dir: String): Array -> Lists the contents of a directory
mkdirs(dir: String): Boolean -> Creates the given directory if it does not exist, also creating any necessary parent
directories
put(file: String, contents: String, overwrite: Boolean = false): Boolean -> Writes the given String out to a file, encoded
in UTF-8
head(file: String, maxBytes: int = 1024 * 100): String -> Returns up to the first 'maxBytes' bytes of the given file as a
String encoded in UTF-8
append(file: String, content: String, createFileIfNotExists: Boolean): Boolean -> Append the content to a file
rm(dir: String, recurse: Boolean = false): Boolean -> Removes a file or directory
exists(file: String): Boolean -> Check if a file or directory exists
mount(source: String, mountPoint: String, extraConfigs: Map[String, Any]): Boolean -> Mounts the given remote storage
directory at the given mount point
unmount(mountPoint: String): Boolean -> Deletes a mount point
mounts(): Array[MountPointInfo] -> Show information about what is mounted
getMountPath(mountPoint: String, scope: String = ""): String -> Gets the local path of the mount point

Use mssparkutils.fs.help("methodName") for more info about a method.
```

mssparkutils works with the file system in the same way as Spark APIs. Take *mssparkuitls.fs.mkdirs()* and Fabric Lakehouse usage for example:

Usage	Relative path from HDFS root	Absolute path for ABFS file system
Nondefault lakehouse	Not supported	*mssparkutils.fs.mkdirs("abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<new_dir>"*
Default lakehouse	Directory under "Files" or "Tables": *mssparkutils.fs.mkdirs("Files/<new_dir>")*	*mssparkutils.fs.mkdirs("abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<new_dir>"*

List files

List the content of a directory, use *mssparkutils.fs.ls('Your directory path')*, for example:

Python

```
mssparkutils.fs.ls("Files/tmp") # works with the default lakehouse files using relative path
mssparkutils.fs.ls("abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<path>")  # based on ABFS file system
mssparkutils.fs.ls("file:/tmp")  # based on local file system of driver node
```

View file properties

Returns file properties including file name, file path, file size, and whether it's a directory and a file.

Python

```
files = mssparkutils.fs.ls('Your directory path')
for file in files:
    print(file.name, file.isDir, file.isFile, file.path, file.size)
```

Create new directory

Creates the given directory if it doesn't exist and any necessary parent directories.

Python

```
mssparkutils.fs.mkdirs('new directory name')
mssparkutils.fs. mkdirs("Files/<new_dir>")  # works with the default lakehouse files using relative path
mssparkutils.fs.ls("abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<new_dir>")  # based on ABFS file system
mssparkutils.fs.ls("file:/<new_dir>")  # based on local file system of driver node
```

Copy file

Copies a file or directory. Supports copy across file systems.

Python

```
mssparkutils.fs.cp('source file or directory', 'destination file or directory', True)# Set the third parameter as True to copy all files and directories recursively
```

Preview file content

Returns up to the first 'maxBytes' bytes of the given file as a String encoded in UTF-8.

Python

```
mssparkutils.fs.head('file path', maxBytes to read)
```

Move file

Moves a file or directory. Supports move across file systems.

Python

```
mssparkutils.fs.mv('source file or directory', 'destination directory', True) # Set the last parameter as True to firstly create the parent directory if it does not exist
```

Write file

Writes the given string out to a file, encoded in UTF-8. Writes the given string out to a file, encoded in UTF-8.

Python

```
mssparkutils.fs.put("file path", "content to write", True) # Set the last parameter as True to overwrite the file if it existed already
```

Append content to a file

Appends the given string to a file, encoded in UTF-8.

Python

```
mssparkutils.fs.append("file path", "content to append", True) # Set the last parameter as True to create the file if it does not exist
```

Delete file or directory

Removes a file or directory.

Python

```
mssparkutils.fs.rm('file path', True) # Set the last parameter as True to remove all files and directories recursively
```

Mount/unmount directory

You can find the detailed usage in File mount and unmount.

Notebook utilities

Use the MSSparkUtils Notebook Utilities to run a notebook or exit a notebook with a value. Run the following command to get an overview of the available methods:

Python

```
mssparkutils.notebook.help()
```

Output:

Console

```
exit(value: String): void -> This method lets you exit a notebook with a value.
run(path: String, timeoutSeconds: int, arguments: Map): String -> This method runs a notebook and returns its exit value.
```

Reference a notebook

Reference a notebook and returns its exit value. You can run nesting function calls in a notebook interactively or in a pipeline. The notebook being referenced runs on the Spark pool of which notebook calls this function.

Python

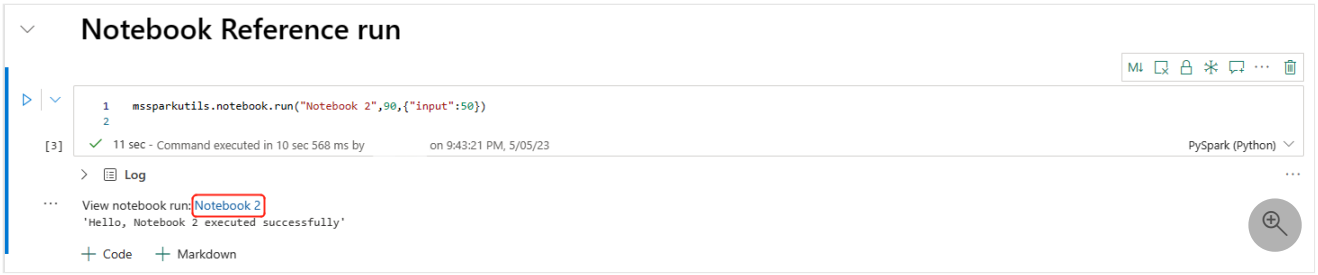
```
mssparkutils.notebook.run("notebook name", <timeoutSeconds>, <parameterMap>)
```

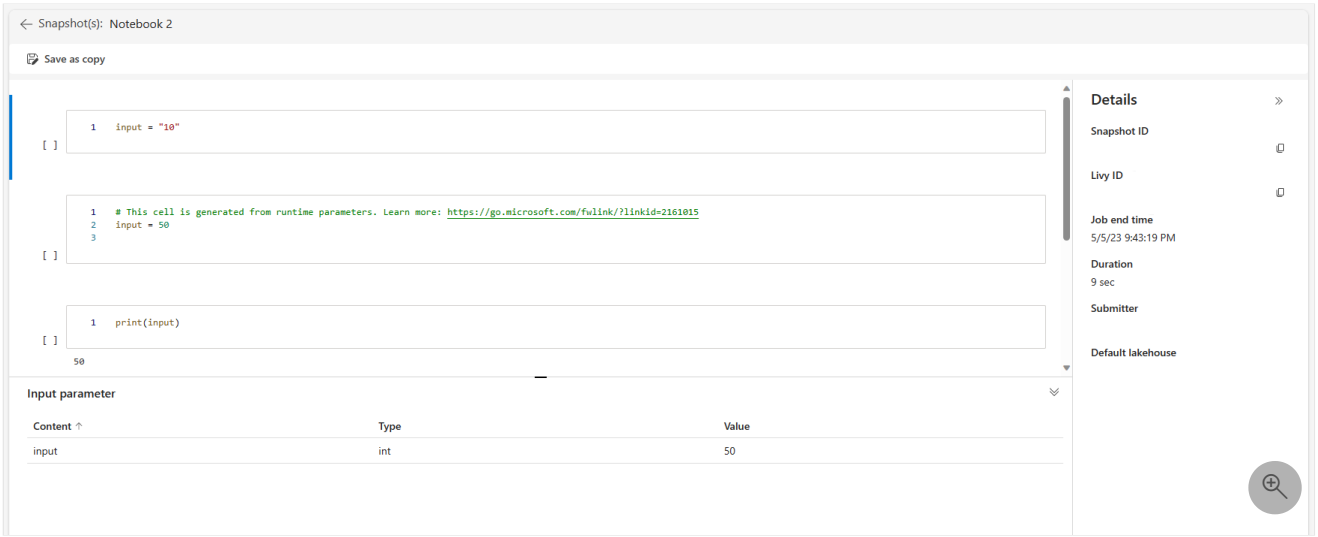
For example:

Python

```
mssparkutils.notebook.run("Sample1", 90, {"input": 20 })
```

You can open the snapshot link of reference run in the cell output, the snapshot captures the code run results and allows you to easily debug a reference run.





> **Note**
>
> Currently Fabric notebook only supports referencing notebooks within a workspace.

Exit a notebook

Exits a notebook with a value. You can run nesting function calls in a notebook interactively or in a pipeline.

- When you call an *exit()* function from a notebook interactively, Fabric notebook throws an exception, skip running subsequence cells, and keep the Spark session alive.
- When you orchestrate a notebook in pipeline that calls an *exit()* function, the Notebook activity will return with an exit value, complete the pipeline run and stop the Spark session.
- When you call an *exit()* function in a notebook that is being referenced, Fabric Spark will stop the further execution of the referenced notebook, and continue to run next cells in the main notebook that calls the *run()* function. For example: Notebook1 has three cells and calls an *exit()* function in the second cell. Notebook2 has five cells and calls *run(notebook1)* in the third cell. When you run Notebook2, Notebook1 stops at the second cell when hitting the *exit()* function. Notebook2 continues to run its fourth cell and fifth cell.

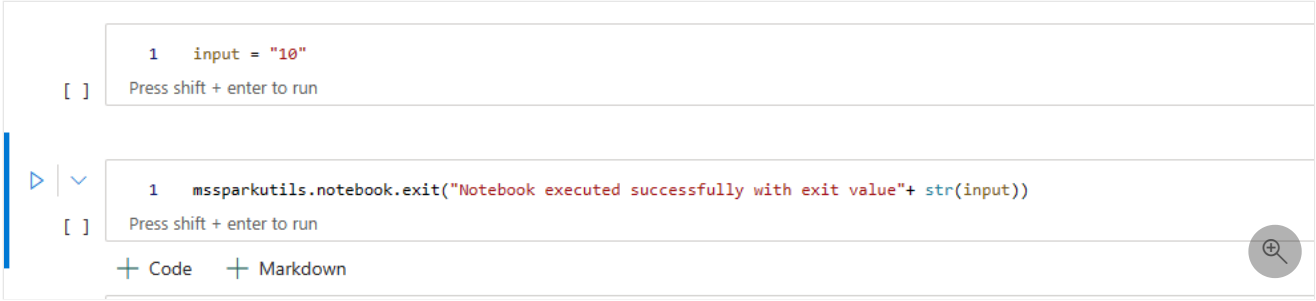
Python

```
mssparkutils.notebook.exit("value string")
```

For example:

Sample1 notebook with following two cells:

- Cell 1 defines an **input** parameter with default value set to 10.
- Cell 2 exits the notebook with **input** as exit value.



You can run the **Sample1** in another notebook with default values:

```
Python

exitVal = mssparkutils.notebook.run("Sample1")
print (exitVal)
```

Output:

```
Console

Notebook executed successfully with exit value 10
```

You can run the **Sample1** in another notebook and set the **input** value as 20:

```
Python

exitVal = mssparkutils.notebook.run("Sample1", 90, {"input": 20 })
print (exitVal)
```

Output:

```
Console

Notebook executed successfully with exit value 20
```

Session management

Stop an interactive session

Instead of manually selecting the stop button, sometimes it's more convenient to stop an interactive session by calling an API in the code. For such cases, we provide an API *mssparkutils.session.stop()* to support stopping the interactive session via code, it's available for Scala and Python.

```
Python

mssparkutils.session.stop()
```

mssparkutils.session.stop() API stops the current interactive session asynchronously in the background, it stops the Spark session and release resources occupied by the session so they're available to other sessions in the same pool.

> ⓘ **Note**
>
> We don't recommend calling language built-in APIs like *sys.exit* in Scala or *sys.exit()* in Python in your code, because such APIs just kill the interpreter process, leaving the Spark session alive and the resources not released.

Credentials utilities

You can use the MSSparkUtils Credentials Utilities to get the access tokens and manage secrets in Azure Key Vault.

Run the following command to get an overview of the available methods:

Python

```
mssparkutils.credentials.help()
```

Output:

Console

```
getToken(audience, name): returns AAD token for a given audience, name (optional)
getSecret(akvName, secret): returns AKV secret for a given akvName, secret key
```

Get token

Returns Azure AD token for a given audience, name (optional), The list below shows currently available audience keys:

- **Storage Audience Resource**: "storage"
- **Power BI Resource**: "pbi"
- **Azure Key Vault Resource**: "keyvault"
- **Synapse RTA KQL DB Resource**: "kusto"

Run the following command to get the token:

Python

```
mssparkutils.credentials.getToken('audience Key')
```

Get secret using user credentials

Returns Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using user credentials.

Python

```
mssparkutils.credentials.getSecret('azure key vault name','secret name')
```

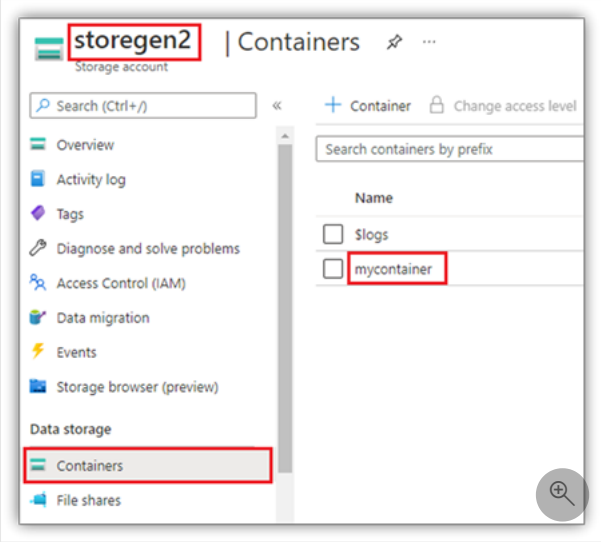
File mount and unmount

The Microsoft Fabric support mount scenarios in the Microsoft Spark Utilities package. You can use *mount*, *unmount*, *getMountPath()* and *mounts()* APIs to attach remote storage (Azure Data Lake Storage Gen2) to all working nodes (driver node and worker nodes). After the storage mount point is in place, use the local file API to access data as if it's stored in the local file system.

How to mount an ADLS Gen2 account

This section illustrates how to mount Azure Data Lake Storage Gen2 step by step as an example. Mounting Blob Storage works similarly.

The example assumes that you have one Data Lake Storage Gen2 account named *storegen2*. The account has one container named *mycontainer* that you want to mount to */test* into your notebook spark session.

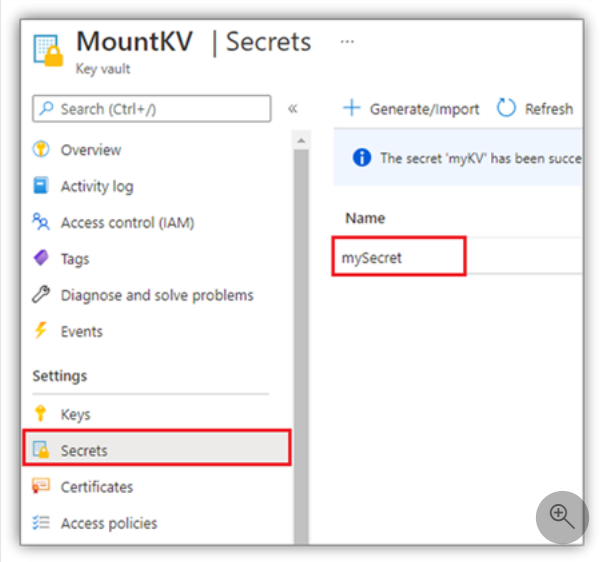


To mount the container called *mycontainer*, *mssparkutils* first needs to check whether you have the permission to access the container. Currently, Microsoft Fabric supports two authentication methods for the trigger mount operation: *accountKey* and *sastoken*.

Mount via shared access signature token or account key

Mssparkutils supports explicitly passing an account key or Shared access signature (SAS) token as a parameter to mount the target.

For security reasons, we recommend that you store account keys or SAS tokens in Azure Key Vault (as the following example screenshot shows). You can then retrieve them by using the *mssparkutils.credentials.getSecret* API. For the usage of Azure Key Vault, refer to About Azure Key Vault managed storage account keys.



Here's the sample code of using accountKey method:

Python

```
from notebookutils import mssparkutils
# get access token for keyvault resource
# you can also use full audience here like https://vault.azure.net
accountKey = mssparkutils.credentials.getSecret("<vaultURI>", "<secretName>")
mssparkutils.fs.mount(
    "abfss://mycontainer@<accountname>.dfs.core.windows.net",
    "/test",
    {"accountKey":accountKey}
)
```

For *sastoken*, reference the following sample code:

Python

```
from notebookutils import mssparkutils
# get access token for keyvault resource
```

```
# you can also use full audience here like https://vault.azure.net
sasToken = mssparkutils.credentials.getSecret("<vaultURI>", "<secretName>")
mssparkutils.fs.mount(
    "abfss://mycontainer@<accountname>.dfs.core.windows.net",
    "/test",
    {"sasToken":sasToken}
)
```

> ⓘ **Note**
>
> For security reasons, it's not recommended to store credentials in code. To further protect your credentials, we will redact your secret in notebook output, for more details please check **Secret redaction**.

How to mount a lakehouse

Here's the sample code of mounting a lakehouse to */test*.

Python

```
from notebookutils import mssparkutils
mssparkutils.fs.mount(
 "abfss://<workspace_id>@msit-onelake.dfs.fabric.microsoft.com/<lakehouse_id>",
 "/test"
)
```

Access files under the mount point by using the *mssparktuils fs* API

The main purpose of the mount operation is to let customers access the data stored in a remote storage account by using a local file system API. You can also access the data by using the *mssparkutils fs* API with a mounted path as a parameter. The path format used here's a little different.

Assume that you mounted the Data Lake Storage Gen2 container *mycontainer* to */test* by using the mount API. When you access the data by using a local file system API, the path format is like this:

Python

```
/synfs/notebook/{sessionId}/test/{filename}
```

When you want to access the data by using the mssparkutils fs API, we recommend using a *getMountPath()* to get the accurate path:

Python

```
path = mssparkutils.fs.getMountPath("/test")
```

- List directories:

  Python

  ```
  mssparkutils.fs.ls(f"file://{mssparkutils.fs.getMountPath('/test')}")
  ```

- Read file content:

  Python

  ```
  mssparkutils.fs.head(f"file://{mssparkutils.fs.getMountPath('/test')}/myFile.txt")
  ```

- Create a directory:

  Python

  ```
  mssparkutils.fs.mkdirs(f"file://{mssparkutils.fs.getMountPath('/test')}/newdir")
  ```

Access files under the mount point via local path

You can easily read and write the files in mount point using standard file system way, use Python as an example:

Python

```
#File read
with open(mssparkutils.fs.getMountPath('/test2') + "/myFile.txt", "r") as f:
    print(f.read())
#File write
with open(mssparkutils.fs.getMountPath('/test2') + "/myFile.txt", "w") as f:
    print(f.write("dummy data"))
```

How to check existing mount points

You can use *mssparkutils.fs.mounts()* API to check all existing mount point info:

Python

```
mssparkutils.fs.mounts()
```

How to unmount the mount point

Use the following code to unmount your mount point *(/test* in this example):

Python

```
mssparkutils.fs.unmount("/test")
```

Known limitations

- The current mount is a job level configuration, we recommand to use *mounts* API to check if mount point exists or not available.
- The unmount mechanism isn't automatic. When the application run finishes, to unmount the mount point to release the disk space, you need to explicitly call an unmount API in your code. Otherwise, the mount point will still exist in the node after the application run finishes.
- Mounting an ADLS Gen1 storage account isn't supported.

Next steps

- Library management

Manage Apache Spark libraries in Microsoft Fabric

Article • 05/23/2023

Libraries provide reusable code that Apache Spark developers may want to include in their Spark application.

Each workspace comes with a pre-installed set of libraries available in the Spark run-time and available to be used immediately in the notebook or Spark job definition. We refer to these as built-in libraries.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Based on the user scenarios and specific needs, you can include other libraries. There are two types of libraries you may want to include:

- **Feed library**: Feed libraries are the ones that come from public sources or repositories. You can install Python feed libraries from PyPI and Conda by specifying the source in the Library Management portals. You can also use a Conda environment specification *.yml* file to install libraries.
- **Custom library**: Custom libraries are the code built by you or your organization. *.whl*, *.jar* and *.tar.gz* can be managed through Library Management portals. Note that *.tar.gz* is only supported for R language, please use *.whl* for Python custom libraries.

Summary of library management and best practices

You can manage all the previously mentioned types of libraries via two different entry points: library management in workspace settings and in-line installation.

1. Workspace library management: Workspace library settings define the working environment for the entire Workspace. The libraries installed on a Workspace level are available for all Notebooks and Spark job definitions under that workspace.

Update the workspace libraries when you want to set up the shared environment for all items in a workspace.

> **ⓘ Important**
>
> Workspace library management is restricted to workspace admin only. Workspace member, contributor and viewer can view the libraries installed by the administrator.

2. In-line installation: With in-line installation, you can install libraries for your Notebook session without affecting the global environment. This is a convenient option when you want a temporary and fast solution. For instance, you might want to try out a local package or use some additional packages for a single session. Currently, Python packages and R packages can be managed in-line.

   > **ⓘ Important**
   >
   > In-line installation is session-specific and does not persist across sessions.
   >
   > The Python interpreter will be restarted to apply the changes of library, any variables defined before running the command cell will be lost. Therefore, we strongly recommend you to put all the commands for adding, deleting, or updating Python packages at the beginning of your Notebook.

Summarizing all library management behaviors currently available in Microsoft Fabric:

Library name	Workspace update	In-line installation
Python Feed (PyPI & Conda)	Supported	Supported
Python Custom (.whl)	Supported	Supported
R Feed (CRAN)	Not Supported	Supported
R custom (.tar.gz)	Supported	Supported
Jar	Supported	Not Supported

> **ⓘ Important**
>
> We currently have limitations of *.jar* library.

- If you upload a *.jar* file with different version of built-in library, it will not be effective on the Starter pool, since all *.jar* files are pre-imported in the Starter pool. Only the new *.jar* will be effective on the Starter pools. The custom spark pools has no such constraints, the custom *.jar* files uploaded with different version will override the built-in ones, and the new ones are also effective.
- *%% configure* magic commands are not fully supported on Fabric at this moment. Please don't use it to bring *.jar* file to your Notebook session.

Library Management in Workspace setting

Under the **Workspace settings**, you find the Workspace level library management portal: **Workspace setting** > **Data engineering** > **Library management**.

Manage feed library in Workspace setting

In this section, we explain how to manage feed libraries from PyPI or Conda using the Workspace library management portal.

- **View and search feed library**: You can see the installed libraries and their name, version, and dependencies on the **library management portal**. You can also use the filter box on the upper right corner to find an installed library quickly.
- **Add new feed library**: The default source for installing Python feed libraries is PyPI. You can also select "Conda" from the drop-down button next to the add button. To add a new library, click on the + button and enter the library name and version in the new row.

  Alternatively, you can upload a .yml file to install multiple feed libraries at once.
- **Remove existing feed library**: To remove a library, click on the Trash button on its row.
- **Update the version of existing feed library**: To change the version of a library, select a different one from the drop-down box on its row.
- **Review and apply changes**: You can review your changes in the "Pending changes" panel. You can remove a change by clicking on the **X** button, or discard all changes by clicking on the **Discard** button at the bottom of the page. When you are satisfied with your changes, click on **Apply** to make these changes effective.

Manage custom libraries in Workspace setting

In this section, we explain how to manage your custom packages, such as *.jar*, using the Workspace library management portal.

- **Upload new custom library**: You can upload your custom codes as packages to the Microsoft Fabric runtime through the portal. The library management module will help you resolve potential conflicts and download dependencies in your custom libraries.

  To upload a package, click on the **Upload** button under the **Custom libraries** panel and select a local directory.

- **Remove existing custom library**: You can remove a custom library from the Spark runtime by clicking on the trash button under the **Custom libraries** panel.

- **Review and apply changes**: As with feed libraries, you can review your changes in the **Pending changes** panel and apply them to your Microsoft Fabric Spark environment of the Workspace.

> ⓘ **Note**
>
> For *.whl* packages, the library installation process will download the dependencies from public sources automatically. However, this feature is not available for *.tar.gz* packages. You need to upload the dependent packages of the main *.tar.gz* package manually if there are any.

Cancel update

The library update process may take some time to complete. You have the option to cancel the process and continue editing while it is updating. The **Cancel** button will appear during the process.

Troubleshooting

If the library update process fails, you will receive a notification. You can click on the **View log** button to see the log details and troubleshoot the problem. If you encounter a system error, you can copy the root activity ID and report it to the support team.

In-line installation

If you want to use some additional packages for a quick test in an interactive Notebook run, in-line installation is the most convenient option.

> **Important**
>
> *%pip* is recommended instead of *!pip*. *!pip* is a IPython built-in shell command which has following limitations:
>
> - *!pip* will only install package on driver node without executor nodes.
> - Packages that install through *!pip* will not effect when conflicts with built-in packages or when it's already imported in Notebook.
>
> However, *%pip* will handle all above mentioned scenarios. Libraries installed through *%pip* will be available on both driver & executor nodes and will be still effective even it's already imported.

> **Tip**
>
> - The *%conda install* command usually takes longer than the *%pip install* command to install new Python libraries, because it checks the full dependencies and resolves conflicts. You may want to use *%conda install* for more reliability and stability. You can use *%pip install* if you are sure that the library you want to install does not conflict with the pre-installed libraries in the runtime environment.
> - All available Python in-line commands and its clarifications can be found: **%pip commands** and **%conda commands**

Manage Python feed libraries through in-line installation

In this example, we show you how to use in-line commands to manage libraries. Suppose you want to use *altair*, a powerful visualization library for Python, for a one-time data exploration. And suppose the library is not installed on Workspace. In the following example, we use conda commands to illustrate the steps.

You can use in-line commands to enable *altair* on your Notebook session without affecting other sessions of the Notebook or other items.

1. Run the following commands in a Notebook code cell to install the *altair* library and *vega_datasets*, which contains dataset you can use to visualize:

```python
%conda install altair          # install latest version through conda command
%conda install vega_datasets   # install latest version through conda command
```

The log in the cell output indicates the result of installation.

2. Import the package and dataset by running the following codes in another Notebook cell:

```python
import altair as alt
from vega_datasets import data
```

3. Now you can play around with the session-scoped *altair* library:

```python
# load a simple dataset as a pandas DataFrame
cars = data.cars()
alt.Chart(cars).mark_point().encode(
x='Horsepower',
y='Miles_per_Gallon',
color='Origin',
).interactive()
```

Manage Python custom libraries through in-line installation

You can upload your Python custom libraries to the **File** folder of the lakehouse attached to your notebook. Go to your lakehouse, click on the **...** icon on the **File** folder, and upload the custom library.

After uploading, you can use the following command to install the custom library to your Notebook session:

```python
# install the .whl through pip command
%pip install /lakehouse/default/Files/wheel_file_name.whl
```

Manage R feed libraries through in-line installation

Microsoft Fabric supports *install.packages()*, *remove.packages()* and *devtools::* commands to manage R libraries.

> **Tip**
>
> All available R in-line commands and its clarifications can be found: **install.packages command**, **remove.package command** and **devtools commands**.

Follow this example to walk through the steps of installing an R feed library:

1. Switch working language to "SparkR(R)" in Notebook ribbon.
2. Run the following command in a Notebook cell to install *caesar* library:

```
Python

install.packages("caesar")
```

3. Now you can play around with the session-scoped *caesar* library with Spark job

```
Python

library(SparkR)
sparkR.session()

hello <- function(x) {
library(caesar)
caesar(x)
}
spark.lapply(c("hello world", "good morning", "good evening"), hello)
```

Next steps

- Apache Spark workspace administration settings

Capacity administration settings for Data Engineering and Data Science

Article • 05/23/2023

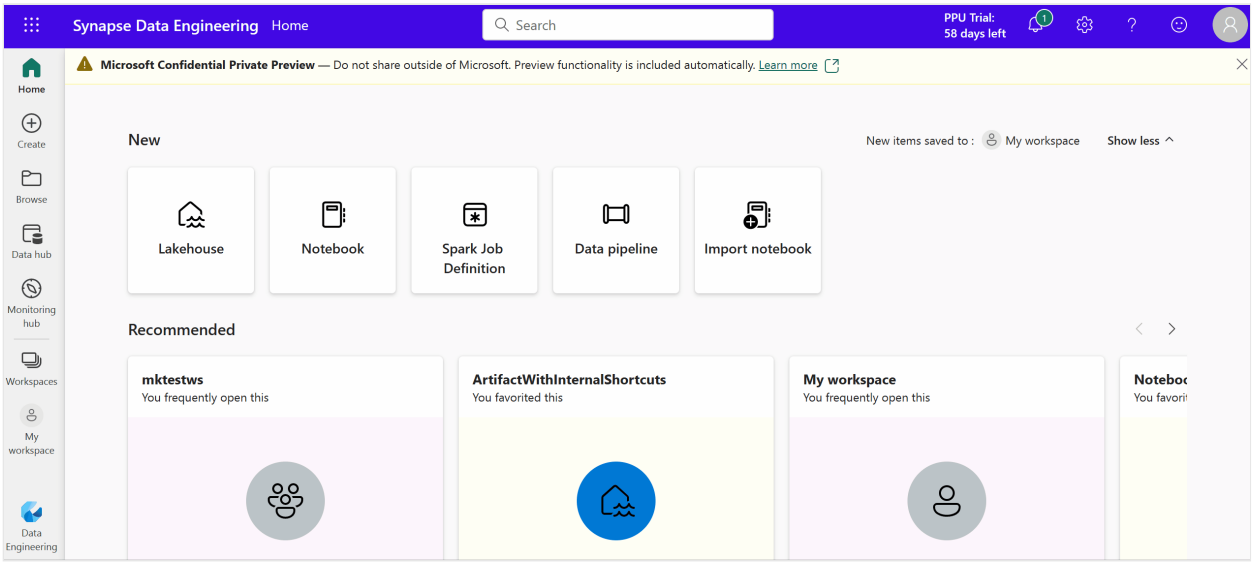
Applies to: ✅ Data Engineering and Data Science in Microsoft Fabric

Admins purchase Microsoft Fabric capacities based on the compute and scale requirements of their enterprise's analytics needs. Admins are responsible to manage the capacity and governance. They must govern and manage the compute properties for data engineering and science analytics applications.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Microsoft Fabric capacity admins can now manage and govern their Data Engineering and Data Science settings from the admin settings portal. Admins can configure Spark environment for their users by enabling workspace level compute, choose a default runtime, and also create or manage spark properties for their capacities.

From the Admin portal, navigate to the **Data Engineering/Science Settings** section and select a specific capacity as shown in the following animation:



Next steps

- Get Started with Data Engineering/Science Admin Settings for your Fabric Capacity

Configure and manage data engineering and data science settings for Fabric capacities

Article • 05/23/2023

Applies to: ✅ Data Engineering and Data Science in Microsoft Fabric

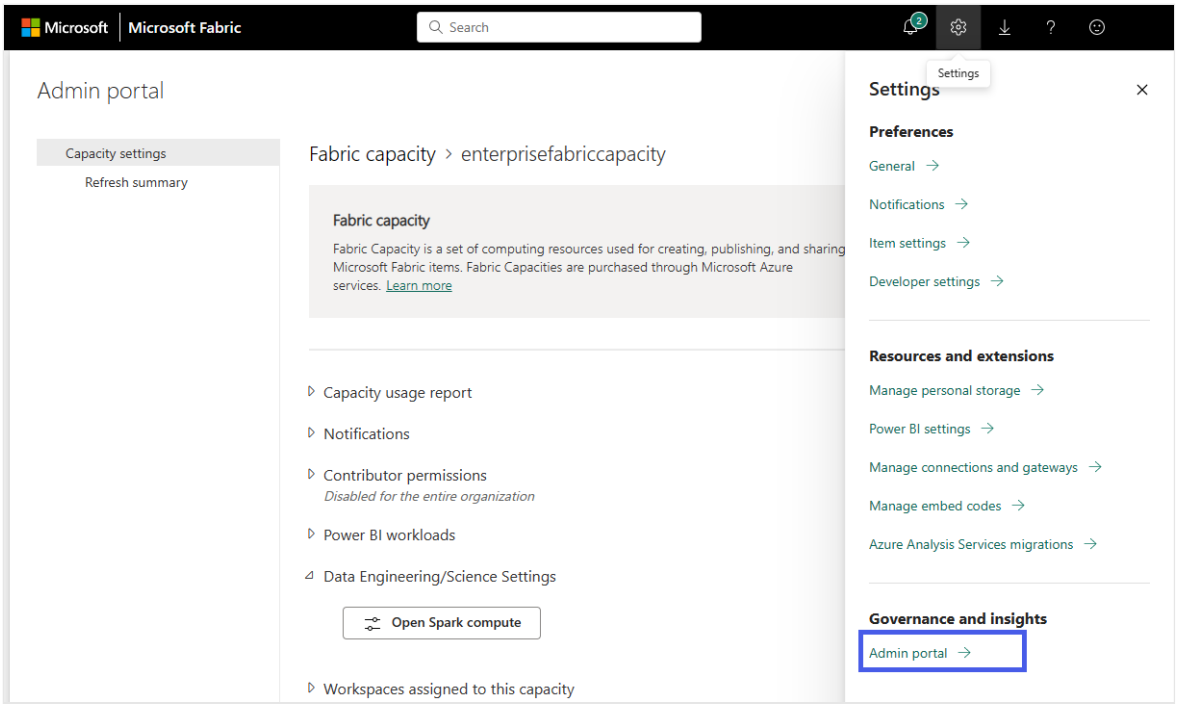
When you create Microsoft Fabric from the Azure portal, it is automatically added to the Fabric tenant that's associated with the subscription used to create the capacity. With the simplified setup in Microsoft Fabric, there's no need to link the capacity to the Fabric tenant. Because the newly created capacity will be listed in the admin settings pane. This configuration provides a faster experience for admins to start setting up the capacity for their enterprise analytics teams.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

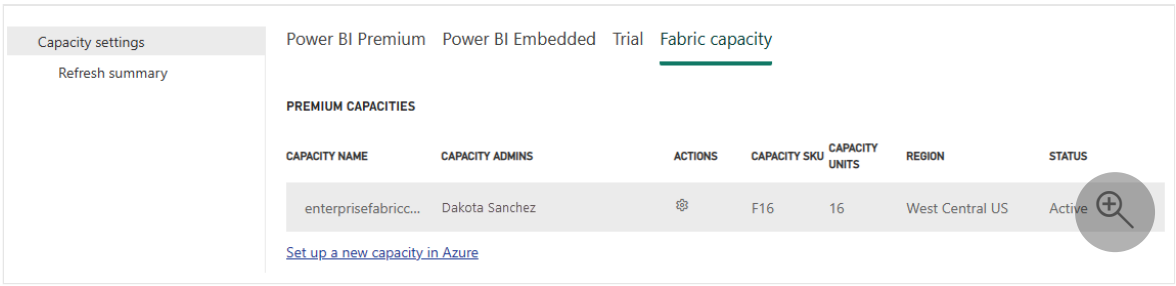
To make changes to the Data Engineering/Science settings in a capacity, you must have admin role for that capacity. To learn more about the roles that you can assign to users in a capacity, see Roles in capacities.

Use the following steps to manage the Data Engineering/Science settings for Microsoft Fabric capacity:

1. Select the **Settings** option to open the setting pane for your Fabric account. Select **Admin portal** under Governance and insights section



2. Choose the **Capacity settings** option to expand the menu and select **Fabric capacity** tab. Here you should see the capacities that you have created in your tenant. Choose the capacity that you want to configure.

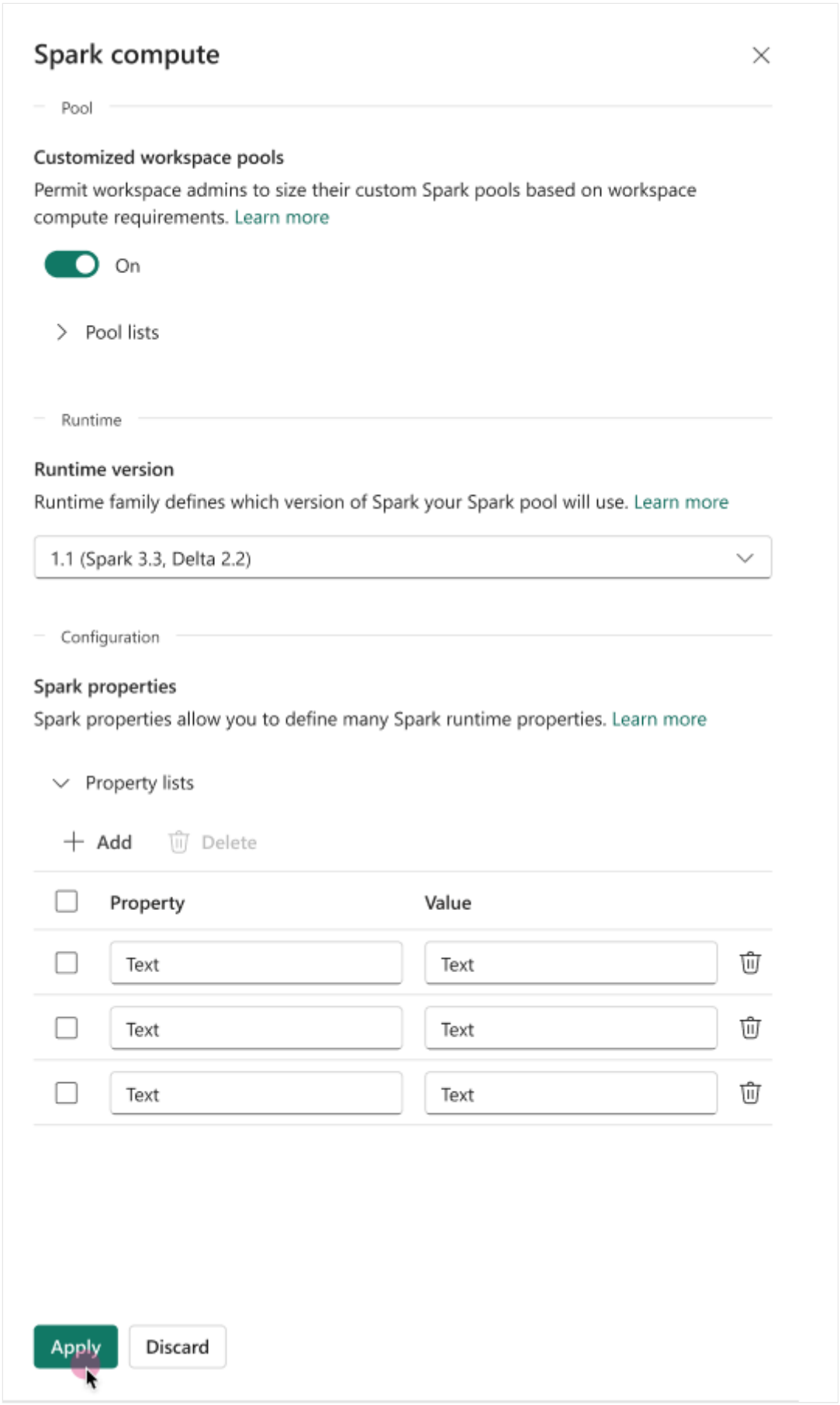


3. You are navigated to the capacities detail pane, where you can view the usage and other admin controls for your capacity. Navigate to the **Data Engineering/Science Settings** section and select **Open Spark Compute**. Configure the following parameters:

    - **Customized workspace pools:** You can restrict or democratize compute customization to workspace admins by enabling or disabling this option. Enabling this option allows workspace admins to create, update, or delete workspace level custom spark pools. Additionally, it allows you to resize them based on the compute requirements within the maximum cores limit of a capacity.

    - **Runtime version:** As a capacity admin, you can select a default runtime version for the entire capacity. All the new workspaces created in that capacity inherit the selected runtime version. Workspace admins can override the default runtime version inherited and choose a different runtime version based on their workspace level requirements.

- **Spark properties:** Capacity admins can configure spark properties and their values, which are inherited to all the workspaces in the capacity. Like the spark runtime version, workspace admins can override these properties for their individual workspaces.



4. After configuring, select **Apply**

Next steps

- Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace
- Learn about the Spark Compute for Fabric Data Engineering/Science experiences

Spark workspace administration settings in Microsoft Fabric

Article • 05/23/2023

Applies to: ✅ Data Engineering and Data Science in Microsoft Fabric

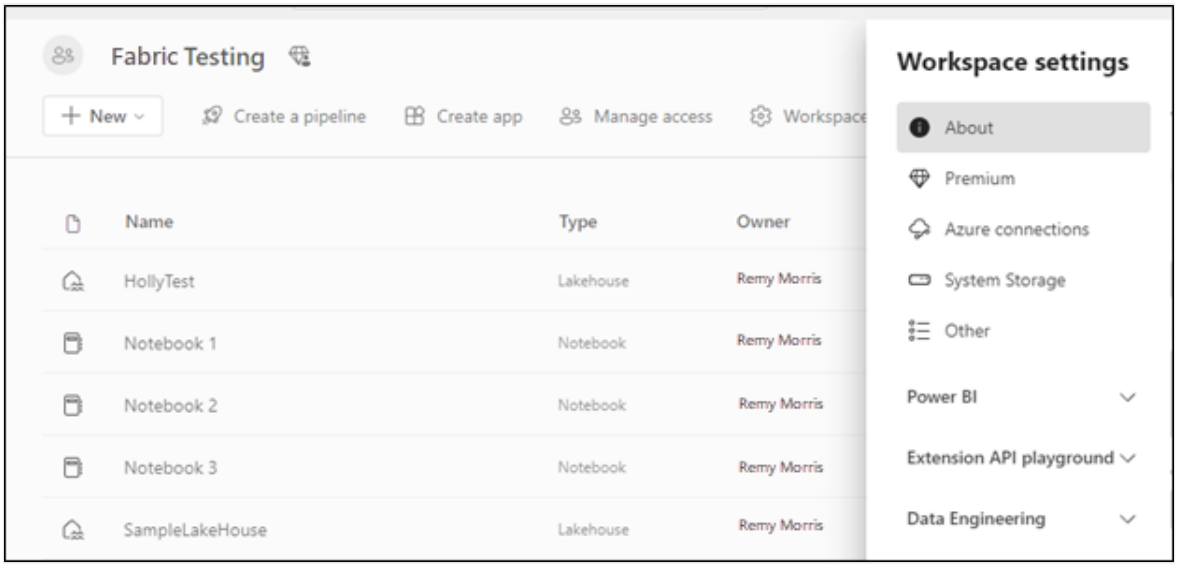
When you create a workspace in Microsoft Fabric, a Starter Pool that is associated with that workspace is automatically created. With the simplified setup in Microsoft Fabric, there's no need to choose the node or machine sizes, as this is handled for you behind the scenes. This configuration provides a faster (5-10 seconds) spark session start experience for users to get started and run your Spark jobs in many common scenarios without having to worry about setting up the compute. For advanced scenarios with specific compute requirements, users can create a custom spark pool and size the nodes based on their performance needs.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

To make changes to the Spark settings in a workspace, you should have the admin role for that workspace. To learn more, see Roles in workspaces.

To manage the Spark settings for the pool associated with your workspace:

1. Go to the **Workspace settings** in your workspace and choose the **Data Engineering/Science** option to expand the menu:



2. You see the **Spark Compute** option in your left-hand menu:

3. Configure the four setting options you can change on this page: **Default pool for workspace**, **Runtime version**, **Automatically track machine learning experiments and models** and **Spark properties**.

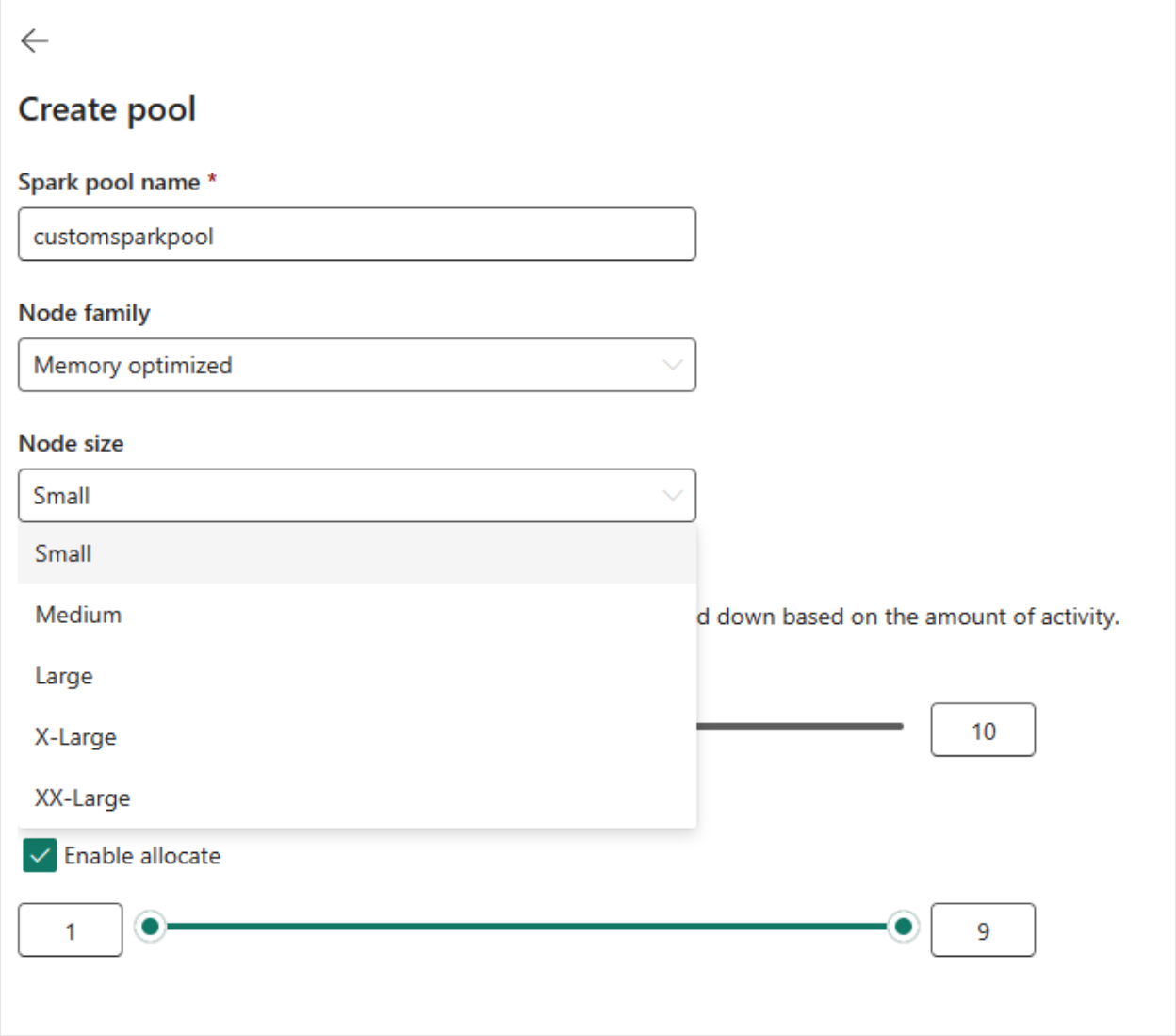
> ⓘ **Note**
>
> If you change the default pool to a custom spark pool you may see longer session start (~3 minutes) in this case.

Default pool for workspace

There are two options:

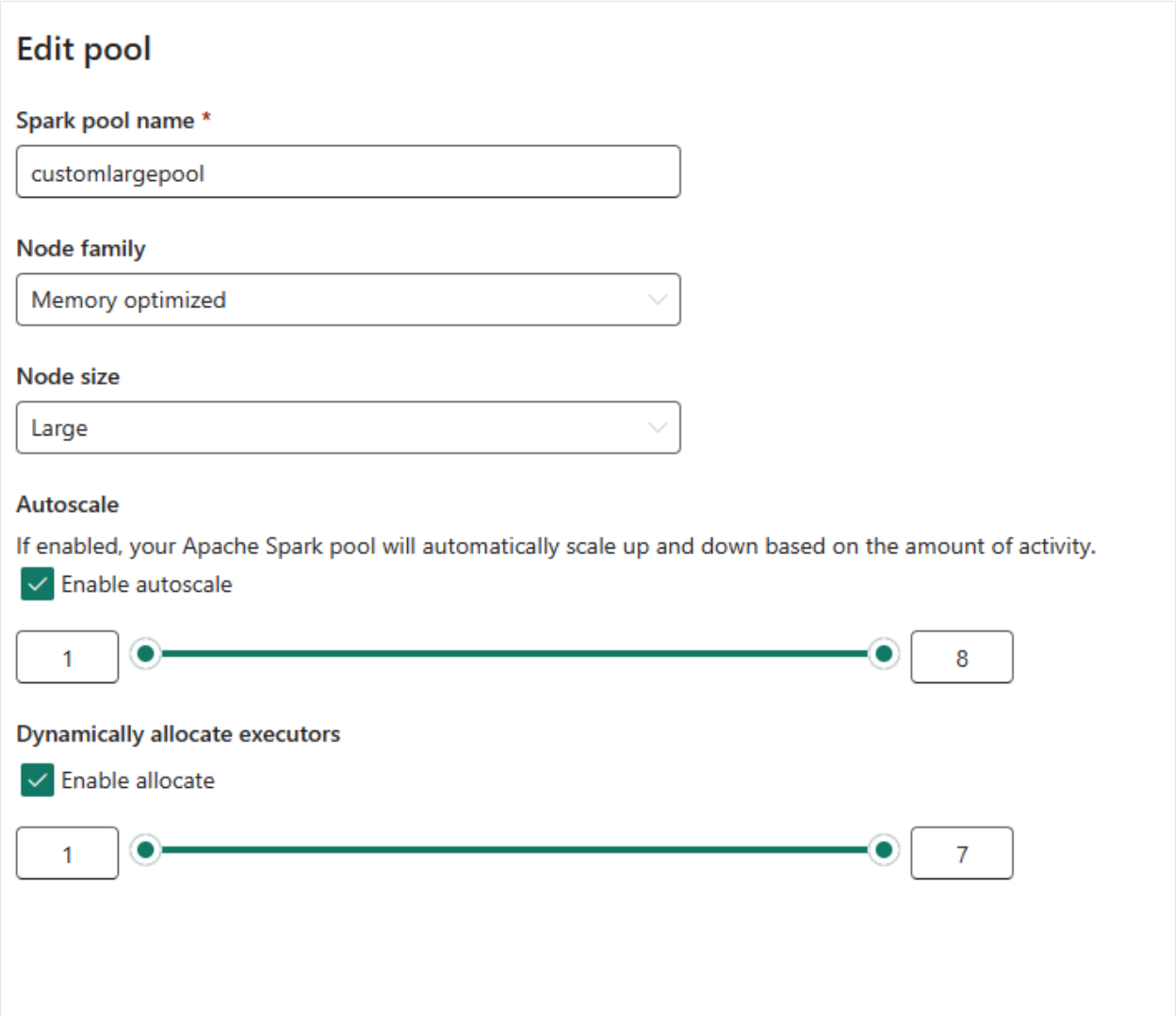
- **Starter Pool**: Prehydrated live clusters automatically created for your faster experience. These are medium size. Currently, a starter pool with 10 nodes is provided for evaluation purposes.
- **Custom Spark Pool**: You can size the nodes, autoscale, and dynamically allocate executors based on your spark job requirements. To create a custom spark pool, the capacity admin should enable the **Customized workspace pools** option in the **Spark Compute** section of **Capacity Admin** settings. To learn more, see Spark Compute Settings for Fabric Capacities.

Admins can create custom spark pools based on their compute requirements by selecting the **New Pool** option.



Microsoft Fabric spark supports single node clusters, which allows users to select a minimum node configuration of 1 and a maximum of 2. Thereby offering high-availability and better job reliability for smaller compute requirements. You can also enable or disable autoscaling option for your custom spark pools. When enabled with autoscale, the pool would acquire new nodes within the max node limit specified by the user and retire them after the job execution for better performance.

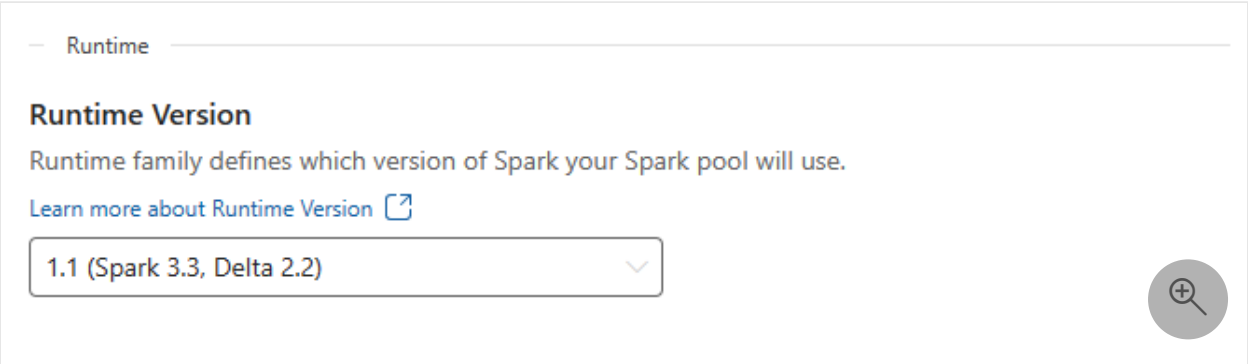
You can also select the option to dynamically allocate executors to pool automatically optimal number of executors within the max bound specified based on the data volume for better performance.



Learn more about Spark Compute for Fabric.

Runtime version

You may choose which version of Spark you'd like to use for the workspace. Currently, Spark 3.2 version is available.



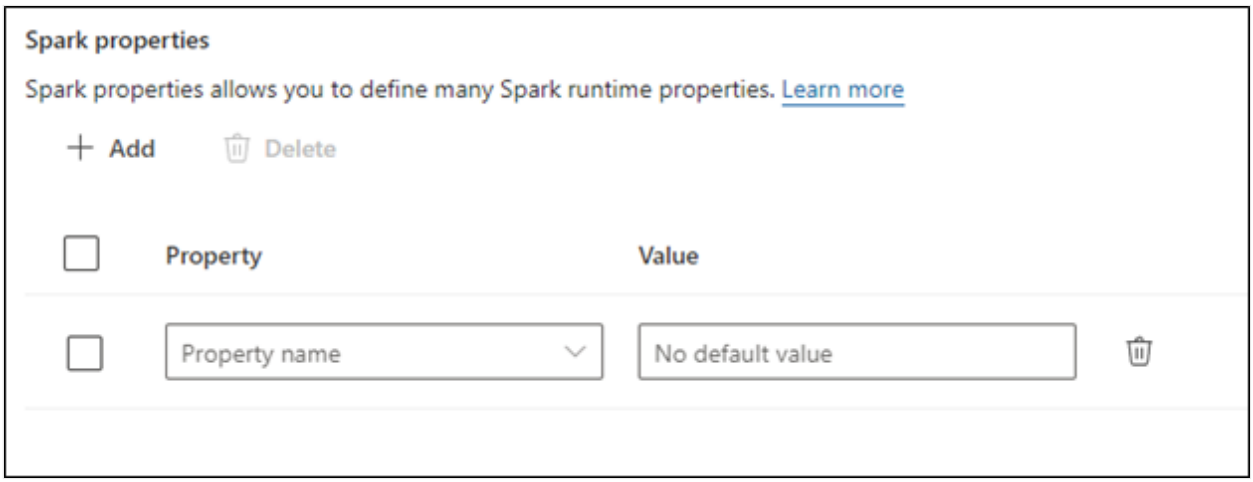
Autologging for Machine Learning models and experiments

Admins can now enable autologging for their machine learning models and experiments. This option will automatically capture the values of input parameters, output metrics, and output items of a machine learning model as it is being trained.

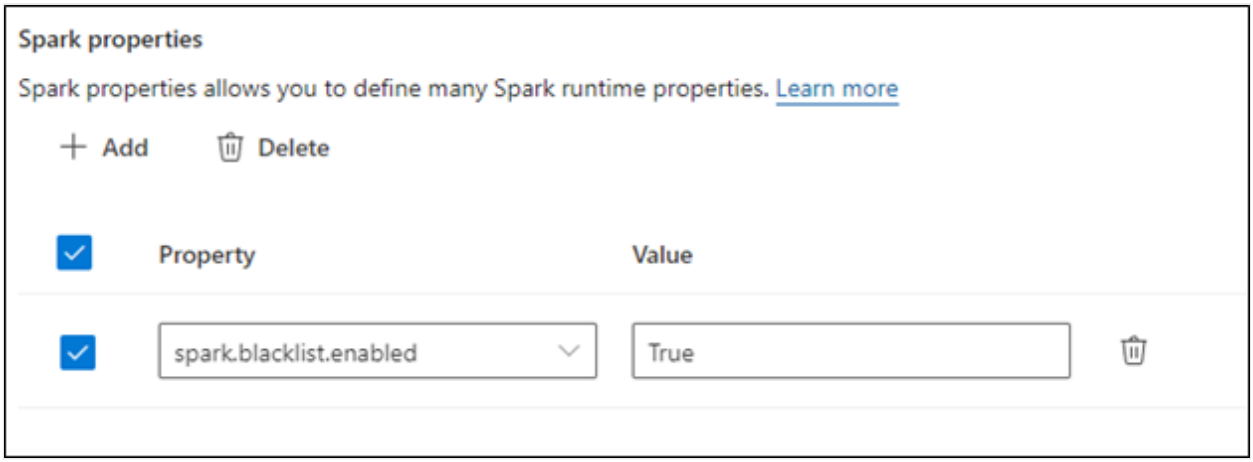
Learn more about autologging

Spark properties

Apache Spark has many settings you can provide to optimize the experience for your scenarios. You may set those properties through the UI by selecting the **Add** option. Select an item from the dropdown menu, and enter the value.



You can delete items by selecting the item(s) and then select the **Delete** button. Or select the delete icon next to each item you wish you to delete.



Next steps

- Learn more from the Apache Spark public documentation.

Apache Spark workspace administration settings FAQ

FAQ

This article lists answers to frequently asked questions about Apache Spark workspace administration settings.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

How do I use the RBAC roles to configure my Spark workspace settings?

Use the **Manage Access** menu to add **Admin** permissions for specific users, distribution groups, or security groups. You can also use this menu to make changes to the workspace and to grant access to add, modify, or delete the Spark workspace settings.

Are the changes made to the Spark properties at the workspace level apply to the active notebook sessions or scheduled Spark jobs?

When you make a configuration change at the workspace level, it's not applied to active Spark sessions. This includes batch or notebook based sessions. You must start a new notebook or a batch session after saving the new configuration settings for the settings to take effect.

Can I configure the node family, Spark runtime, and Spark properties at a capacity level?

Yes, you can change the runtime, or manage the spark properties using the Data Engineering/Science settings as part of the capacity admin settings page. You need have the capacity admin access to view and change these capacity settings.

Can I choose different node families for different notebooks and Spark job definitions in my workspace?

Currently, you can only select Memory Optimized based node family for the entire workspace.

What versions of Spark are supported?

Currently, Spark version 3.3 is the only supported version. Additional versions will be available in the upcoming release.

Can I configure these settings at a notebook level?

Currently, the Spark administration settings are only available at the workspace and capacity level

Can I configure the minimum and maximum number of nodes for the selected node family?

Currently, node limit configuration isn't available. This capability will be enabled in future releases.

Can I enable Autoscaling for the Spark Pools in a memory optimized or hardware accelerated GPU based node family?

Autoscaling isn't currently available. This capability will be enabled in future releases.

Is Intelligent Caching for the Spark Pools supported or enabled by default for a workspace?

Intelligent Caching is enabled by default for the Spark pools for all workspaces.

Microsoft Fabric Apache Spark monitoring overview

Article • 05/23/2023

Microsoft Fabric Spark monitoring is designed to offer a web-UI based experience with built-in rich capabilities for monitoring the progress and status of Spark applications in progress, browsing past Spark activities, analyzing and optimizing performance, and facilitating troubleshooting of failures. Multiple entry points are available for browsing, monitoring, and viewing Spark application details.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Monitoring hub

The Monitoring Hub serves as a centralized portal for browsing Spark activities across items. At a glance, you can view in-progress Spark applications triggered from Notebooks, Spark Job Definitions, and Pipelines. You can also search and filter Spark applications based on different criteria and drill down to view more Spark execution details of a Spark application.

Item recent runs

When working on specific items, the item Recent Runs feature allows you to browse the item's current and recent activities and gain insights on the submitter, status, duration, and other information for activities submitted by you or others.

Notebook contextual monitoring

Notebook Contextual Monitoring offers the capability of authoring, monitoring, and debugging Spark jobs within a single place. You can monitor Spark job progress, view Spark execution tasks and executors, and access Spark logs within a Notebook at the Notebook cell level. The Spark advisor is also built into Notebook to offer real-time advice on code and cell Spark execution and perform error analysis.

Spark job definition inline monitoring

The Spark job definition Inline Monitoring feature allows you to view Spark job definition submission and run status in real-time, as well as view the Spark job definition's past runs and configurations. You can navigate to the Spark application detail page to view more details.

Pipeline Spark activity inline monitoring

For Pipeline Spark Activity Inline Monitoring, deep links have been built into the Notebook and Spark job definition activities within the Pipeline. You can view Spark application execution details, the respective Notebook and Spark job definition snapshot, and access Spark logs for troubleshooting. If the Spark activities fail, the inline error message is also available within Pipeline Spark activities.

Next steps

- Workspace item recent runs
- Notebook contextual monitoring and debugging
- Run an Apache Spark job definition
- Apache Spark application detail monitoring

Apache Spark advisor for real-time advice on notebooks

Article • 05/23/2023

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

The Apache Spark advisor analyzes commands and code run by Apache Spark and displays real-time advice for Notebook runs. The Apache Spark advisor has built-in patterns to help users avoid common mistakes. It offers recommendations for code optimization, performs error analysis, and locates the root cause of failures.

Built-in advice

The Spark advisor, a tool integrated with Impulse, provides built-in patterns for detecting and resolving issues in Apache Spark applications. This article explains some of the patterns included in the tool.

You can open the **Recent runs** pane based on the type of advice you need.

May return inconsistent results when using 'randomSplit'

Inconsistent or inaccurate results may be returned when working with the *randomSplit* method. Use Apache Spark (RDD) caching before using the randomSplit() method.

Method randomSplit() is equivalent to performing sample() on your data frame multiple times. Where each sample refetches, partitions, and sorts your data frame within partitions. The data distribution across partitions and sorting order is important for both randomSplit() and sample(). If either changes upon data refetch, there may be duplicates or missing values across splits. And the same sample using the same seed may produce different results.

These inconsistencies may not happen on every run, but to eliminate them completely, cache your data frame, repartition on a column(s), or apply aggregate functions such as *groupBy*.

Table/view name is already in use

A view already exists with the same name as the created table, or a table already exists with the same name as the created view. When this name is used in queries or applications, only the view will be returned no matter which one created first. To avoid conflicts, rename either the table or the view.

Unable to recognize a hint

Scala

```
spark.sql("SELECT /*+ unknownHint */ * FROM t1")
```

Unable to find a specified relation name(s)

Unable to find the relation(s) specified in the hint. Verify that the relation(s) are spelled correctly and accessible within the scope of the hint.

Scala

```
spark.sql("SELECT /*+ BROADCAST(unknownTable) */ * FROM t1 INNER JOIN t2 ON
t1.str = t2.str")
```

A hint in the query prevents another hint from being applied

The selected query contains a hint that prevents another hint from being applied.

Scala

```
spark.sql("SELECT /*+ BROADCAST(t1), MERGE(t1, t2) */ * FROM t1 INNER JOIN
t2 ON t1.str = t2.str")
```

Enable 'spark.advise.divisionExprConvertRule.enable' to reduce rounding error propagation

This query contains the expression with Double type. We recommend that you enable the configuration 'spark.advise.divisionExprConvertRule.enable', which can help reduce the division expressions and to reduce the rounding error propagation.

```
```

Console

```
"t.a/t.b/t.c" convert into "t.a/(t.b * t.c)"
```

Enable 'spark.advise.nonEqJoinConvertRule.enable' to improve query performance

This query contains time consuming join due to "Or" condition within query. We recommend that you enable the configuration 'spark.advise.nonEqJoinConvertRule.enable', which can help to convert the join triggered by "Or" condition to SMJ or BHJ to accelerate this query.

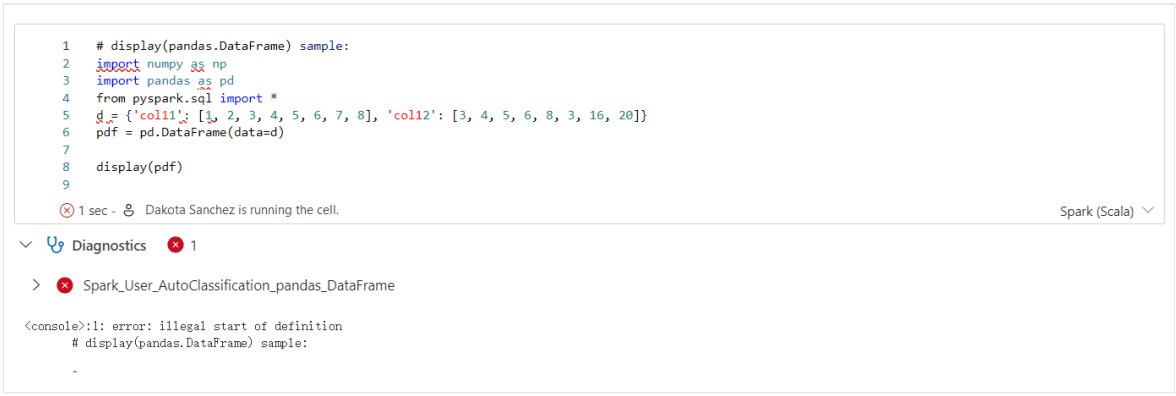
User experience

The Apache Spark advisor displays the advice, including info, warnings, and errors, at Notebook cell output in real-time.

- Info

- Warning

- Error



Next steps

- Monitor Apache Spark jobs within notebooks
- Monitor Apache Spark job definition
- Monitor Apache Spark application details

Browse the Apache Spark applications in the Fabric monitoring hub

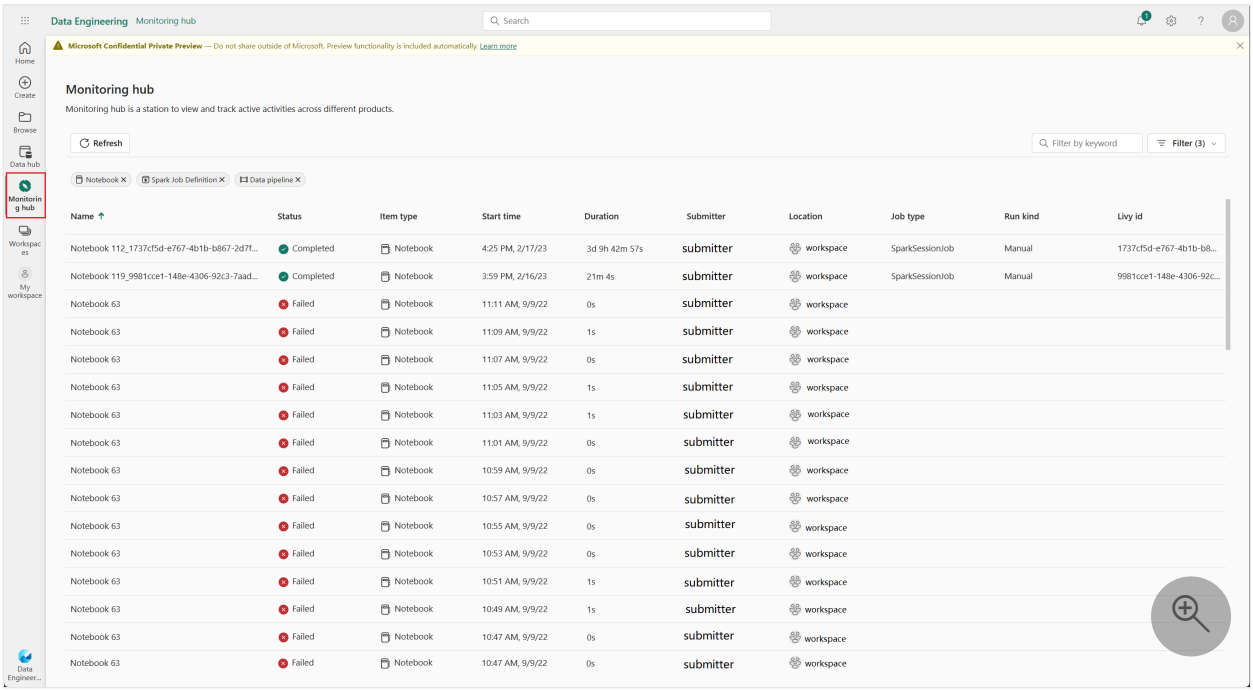
Article • 05/23/2023

The Monitoring hub serves as a centralized portal for browsing Apache Spark activities across items. When you are in the Data Engineering or Data Science experience, you can view in-progress Apache Spark applications triggered from Notebooks, Apache Spark job definitions, and Pipelines. You can also search and filter Apache Spark applications based on different criteria. Additionally, you can cancel your in-progress Apache Spark applications and drill down to view more execution details of an Apache Spark application.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Access the monitoring hub

You can access the Monitoring hub to view various Apache Spark activities by selecting **Monitoring hub** in the left-side navigation links.

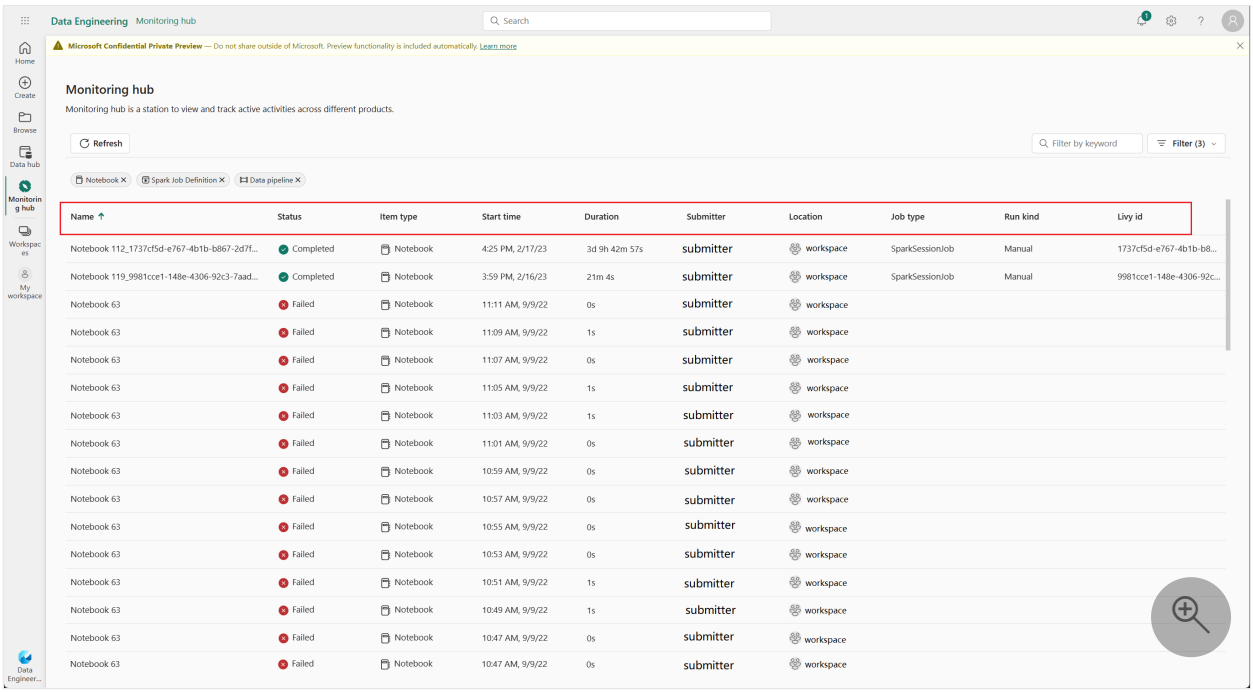


Sort, search and filter Apache Spark applications

For better usability and discoverability, you can sort the Apache Spark applications by selecting different columns in the UI. You can also filter the applications based on different columns and search for specific applications.

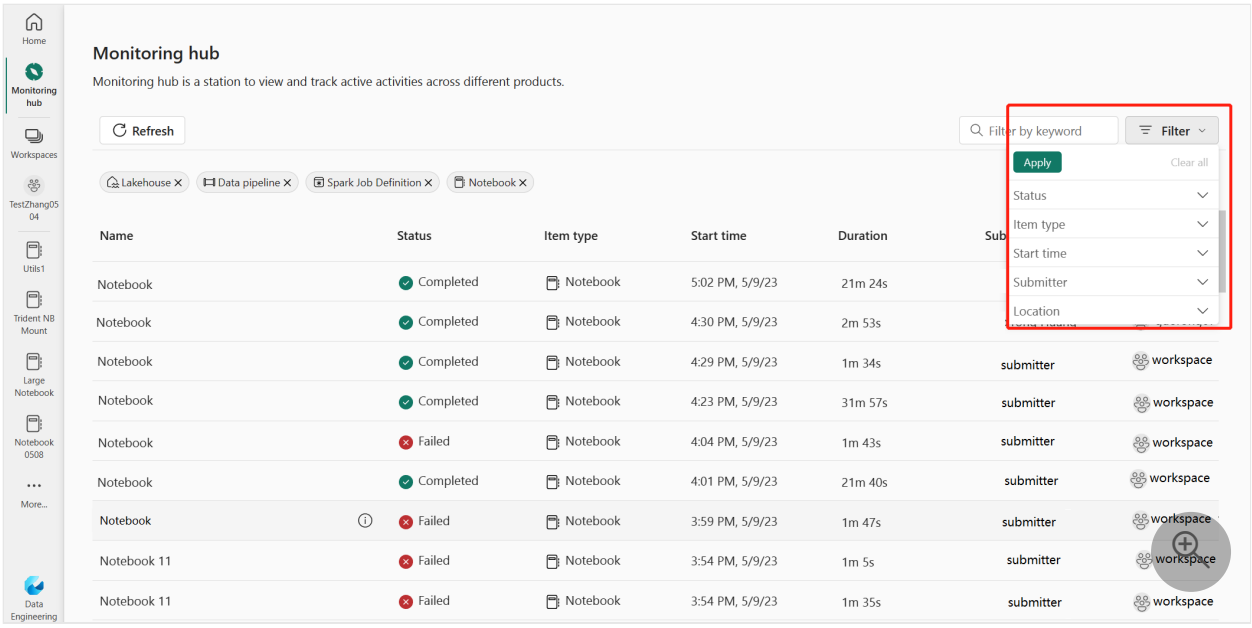
Sort Apache Spark applications

To sort Apache Spark applications, you can select on each column header, such as **Name**, **Status**, **Item Type**, **Start Time**, **Location**, and so on.



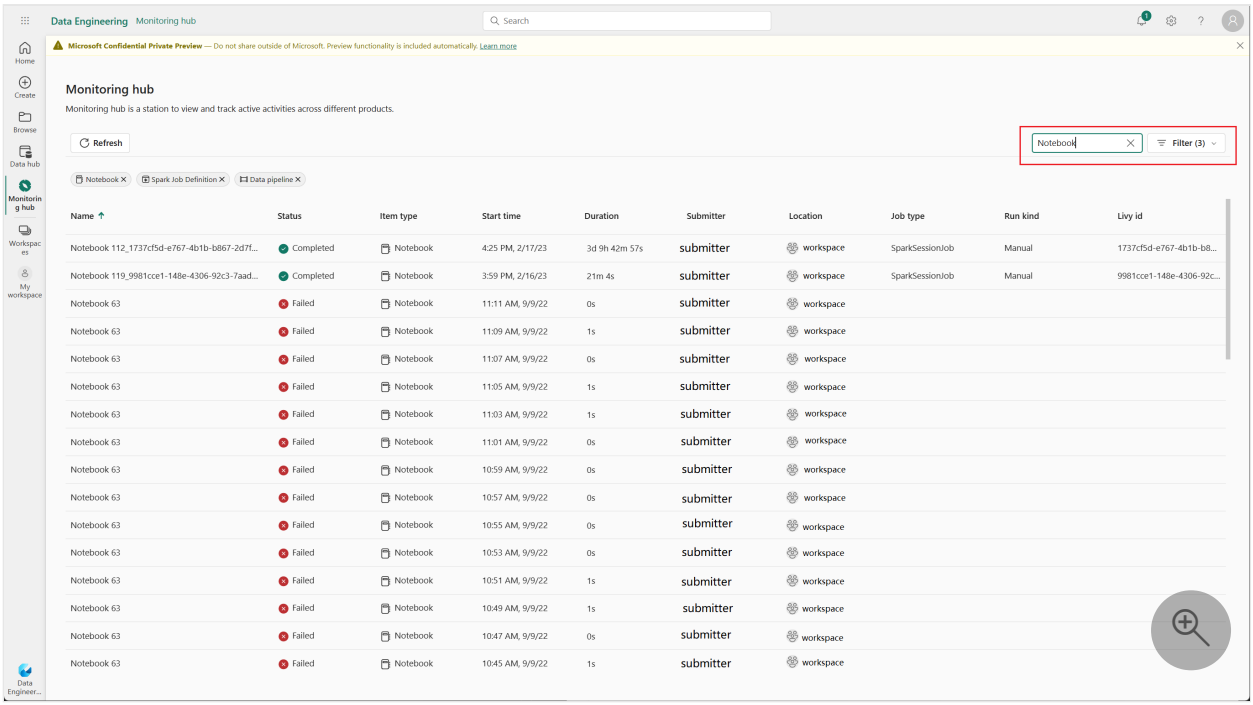
Filter Apache Spark applications

You can filter Apache Spark applications by **Status**, **Item Type**, **Start Time**, **Submitter**, and **Location** using the Filter pane in the upper-right corner.



Search Apache Spark applications

To search for specific Apache Spark applications, you can enter certain keywords in the search box located in the upper-right corner.

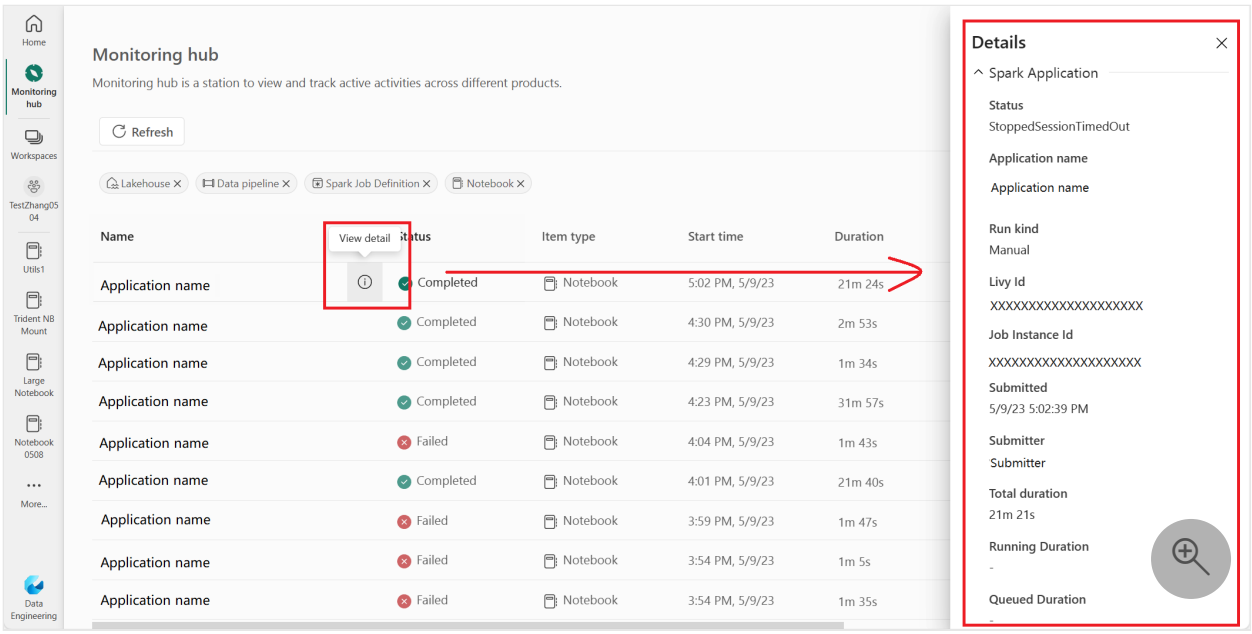


Manage an Apache Spark application

When you hover over an Apache Spark application row, you can see various row-level actions that enable you to manage a particular Apache Spark application.

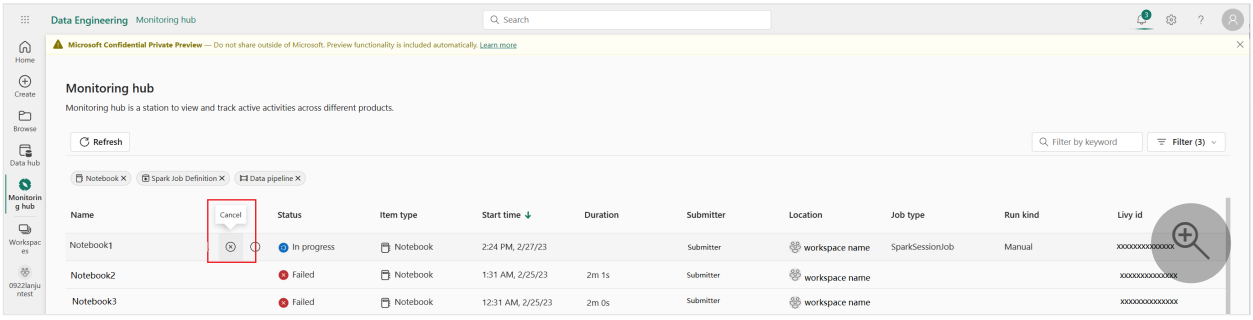
View Apache Spark application detail pane

You can hover over an Apache Spark application row and click the **View details** icon to open the **Detail** pane and view more details about an Apache Spark application.



Cancel an Apache Spark application

If you need to cancel an in-progress Apache Spark application, hover over its row and click the **Cancel** icon.



Navigate to Apache Spark application detail view

If you need more information about Apache Spark execution statistics, access Apache Spark logs, or check input and output data, you can click on the name of an Apache Spark application to navigate to its corresponding Apache Spark application detail page.

Next steps

- Apache Spark monitoring overview
- Browse item's recent runs

- Monitor Apache Spark jobs within notebooks
- Monitor Apache Spark job definition
- Monitor Apache Spark application details

Workspace item recent runs

Article • 05/23/2023

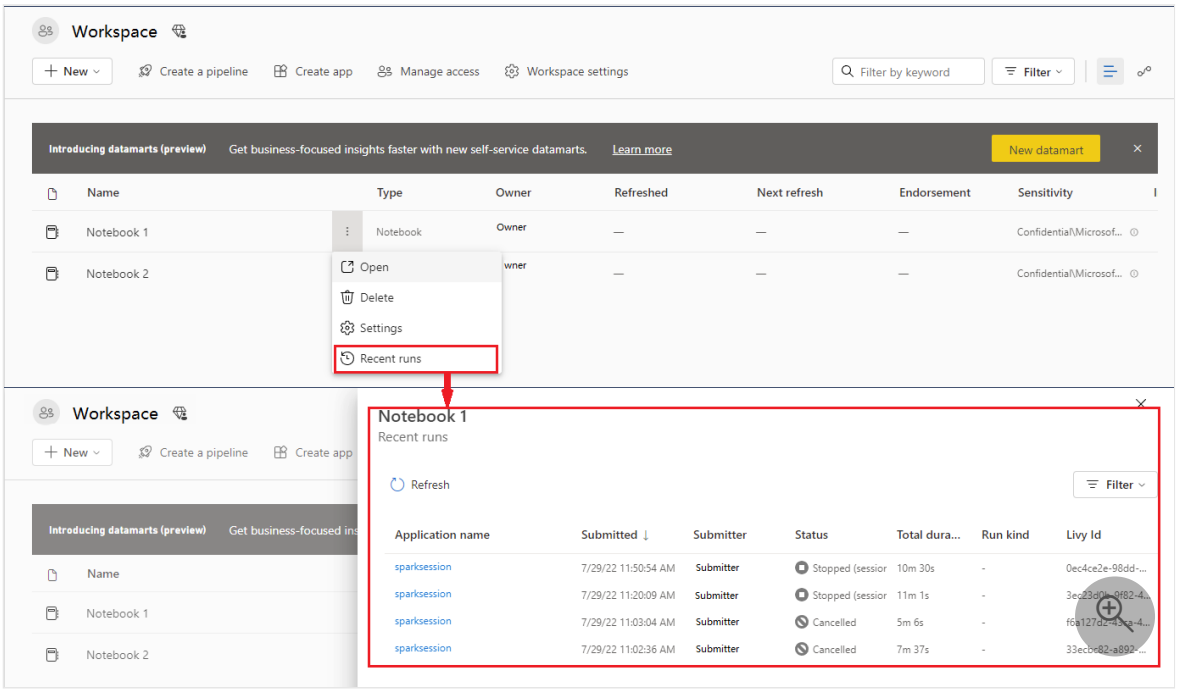
With Microsoft Fabric, you can use Apache Spark to run notebooks, Apache Spark job definitions, jobs, and other types of applications in your workspace. This article explains how to view your running Apache Spark applications, making it easier to keep an eye on the latest running status.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

View the recent runs pane

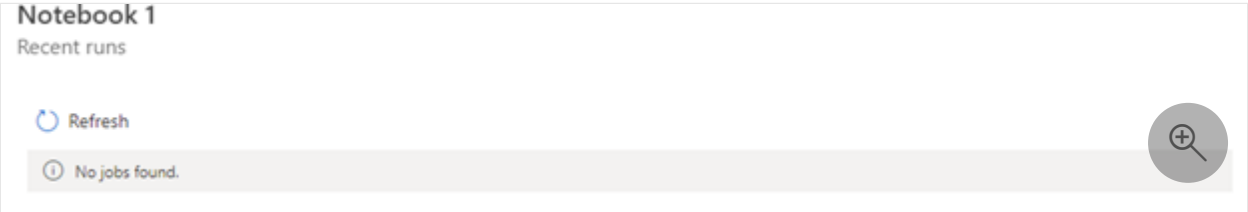
We can open **Recent runs** pane with the following steps:

1. Open the Microsoft Fabric homepage and select a workspace where you want to run the job.
2. Selecting **Spark job definition** or **notebook item context** menu shows the recent run option.
3. Select **Recent runs**.

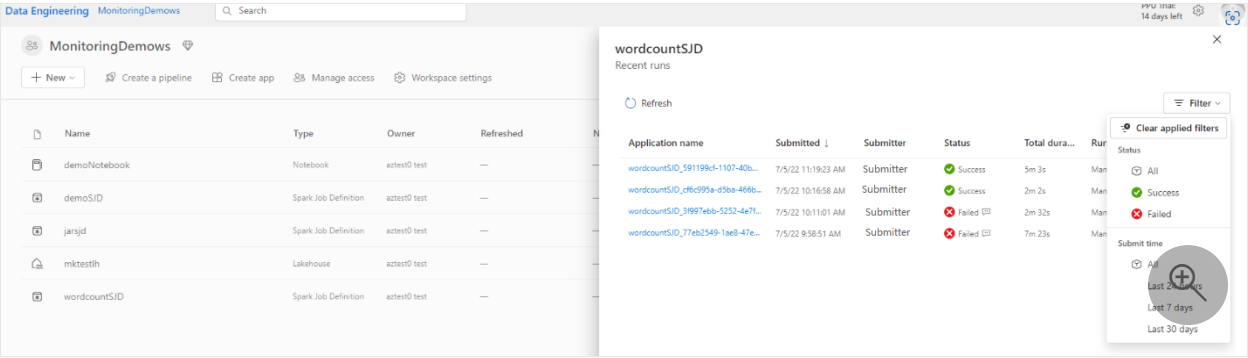


Detail for recent run pane

If the notebook or Spark job definition doesn't have any run operations, the **Recent runs** page shows **No jobs found**.



In the **Recent runs** pane, you can view a list of applications, including **Application name**, **Submitted** time, **Submitter**, **Status**, **Total duration**, **Run kind**, and **Livy Id**. You can filter applications by their status and submission time, which makes it easier for you to view applications.



Selecting the application name link navigates to spark application details where we can get to see the logs, data and skew details for the Spark run.

Next steps

The next step after viewing the list of running Apache Spark applications is to view the application details. You can refer to:

- Apache Spark application detail monitoring

Notebook contextual monitoring & debugging

Article • 05/23/2023

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

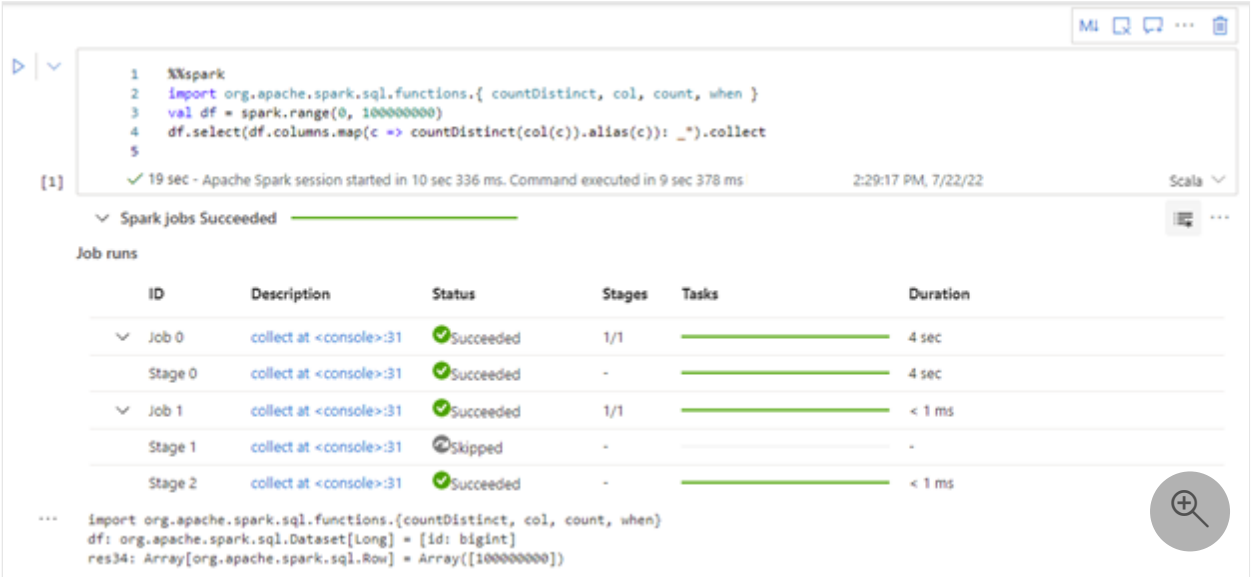
Notebook is purely Apache Spark based. Code cells are executed on the serverless remotely. A Spark job progress indicator is provided with a real-time progress bar appears to help you understand the job execution status.

Monitor Job progress

Run the following sample code and validate the Spark job progress indicator below the Notebook cell.

Python

```
%%spark
import org.apache.spark.sql.functions.{ countDistinct, col, count, when }
val df = spark.range(0, 100000000)
df.select(df.columns.map(c => countDistinct(col(c)).alias(c)): _*).collect
```

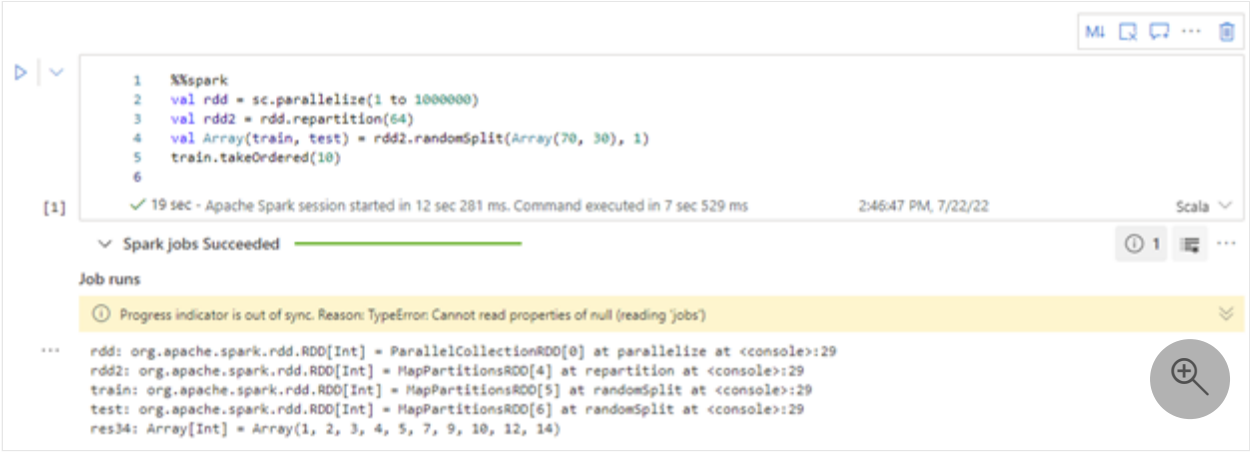


Spark Advisor recommendation

Also supports viewing Spark Advisor info advice, after applying the advice, you would have chance to improve your execution performance, decrease cost and fix the execution failures. Run the sample code below and validate the Spark advisor info message below the Notebook cell.

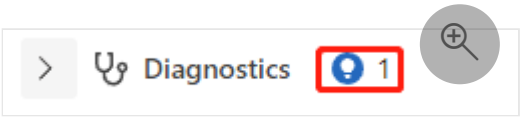
Python

```
%%spark
val rdd = sc.parallelize(1 to 1000000)
val rdd2 = rdd.repartition(64)
val Array(train, test) = rdd2.randomSplit(Array(70, 30), 1)
train.takeOrdered(10)
```

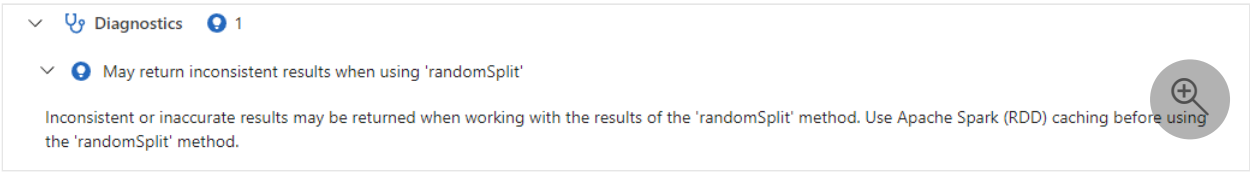


View advice

Icon with blue light bulbs indicate suggestions are available for commands. The box shows the number of different suggestions.



Click the light bulb to expand and view the advice. One or more pieces of advice will become visible.

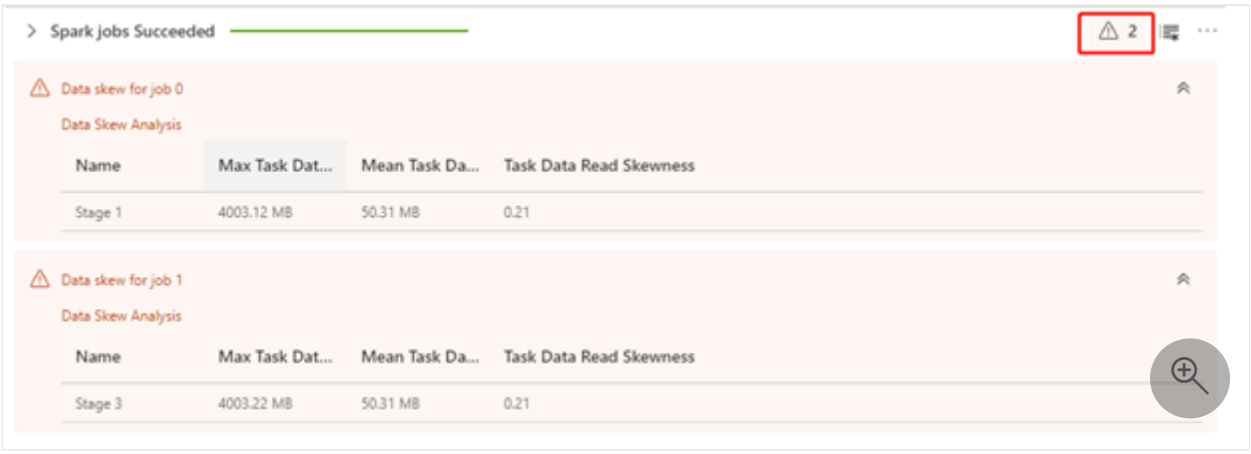


Spark Advisor skew detection

It also supports viewing Spark Advisor skew detection. Run the sample code below and view the Data skew + time skew warning message below the Notebook cell.

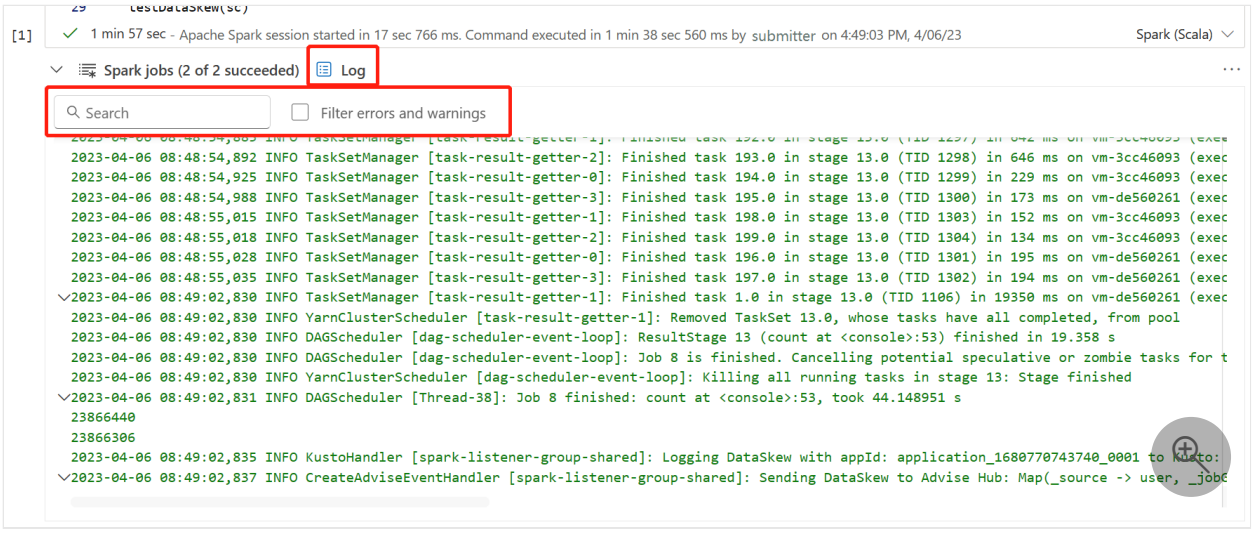
Python

```
%%spark
import org.apache.spark.SparkContext
import java.util.Random
def testDataSkew(sc: SparkContext): Unit = {
    val numMappers = 400
    val numKVPairs = 100000
    val valSize = 256
    val numReducers = 200
    val biasPct = 0.4
    val biasCount = numKVPairs * biasPct
    for (i <- 1 to 2) {
      val query = sc.parallelize(0 until numMappers, numMappers).flatMap { p
=>
        val ranGen = new Random
        val arr1 = new Array[(Int, Array[Byte])](numKVPairs)
        for (i <- 0 until numKVPairs) {
          val byteArr = new Array[Byte](valSize)
          ranGen.nextBytes(byteArr)
          var key = ranGen.nextInt(Int.MaxValue)
          if(i <= biasCount) {
            key = 1
          }
          arr1(i) = (key, byteArr)
        }
        arr1
      }.groupByKey(numReducers)
      println(query.count())
    }
  }
  testDataSkew(sc)
```



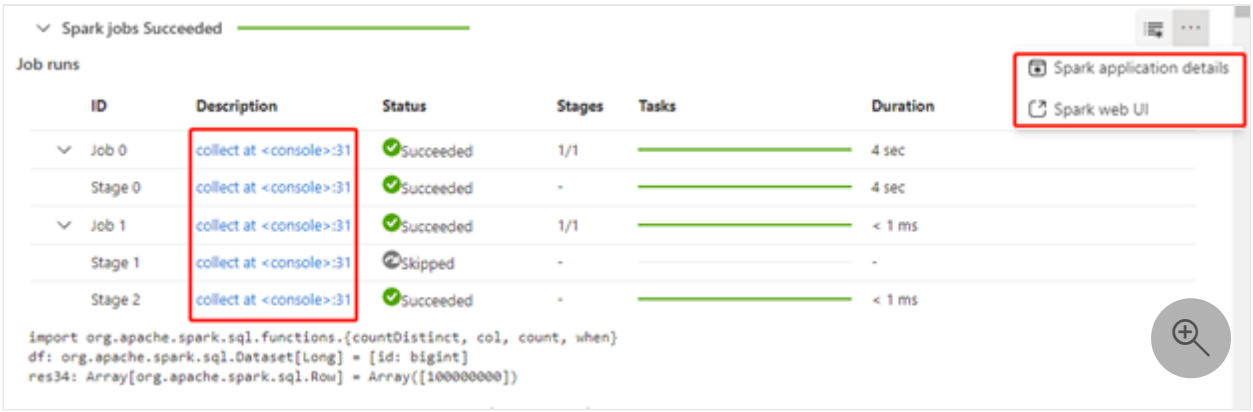
Real-time logs

The log item is shown in cell output, which will display the real-time log. You can search keywords in searchbox or check filter errors and warnings to filter the logs you need.



Access Spark UI and monitoring detail page

The number of tasks per each job or stage helps you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) via selecting the link on the job (or stage) name. And you can also drill down to a specific job (or stage) by selecting the Spark Web UI link in the expand menu to open the Spark History Server.



Next steps

- Spark advisor
- Apache Spark application detail monitoring

Monitor your Apache Spark job definition

Article • 05/23/2023

Using the Spark job definition item's inline monitoring, you can track the following:

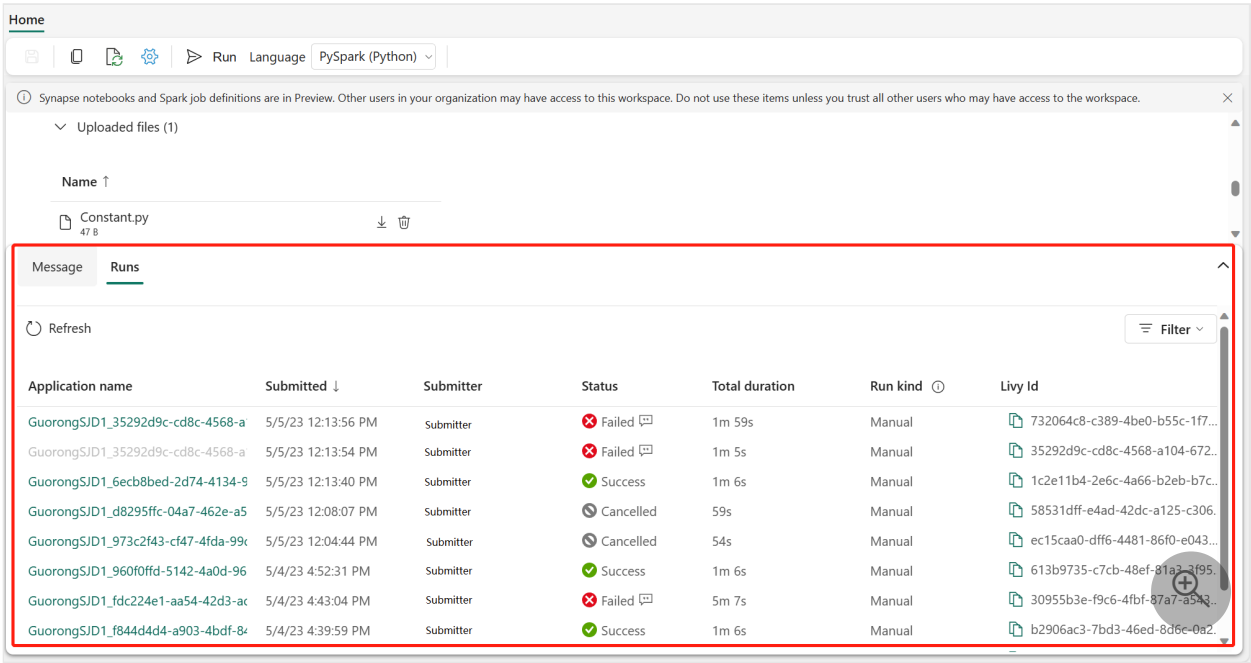
- Monitor the progress and status of a running Spark job definition.
- View the status and duration of previous Spark job definition runs.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

You can get this information from the **Recent Runs** contextual menu in the workspace or by browsing the Spark job definition activities in the monitoring hub.

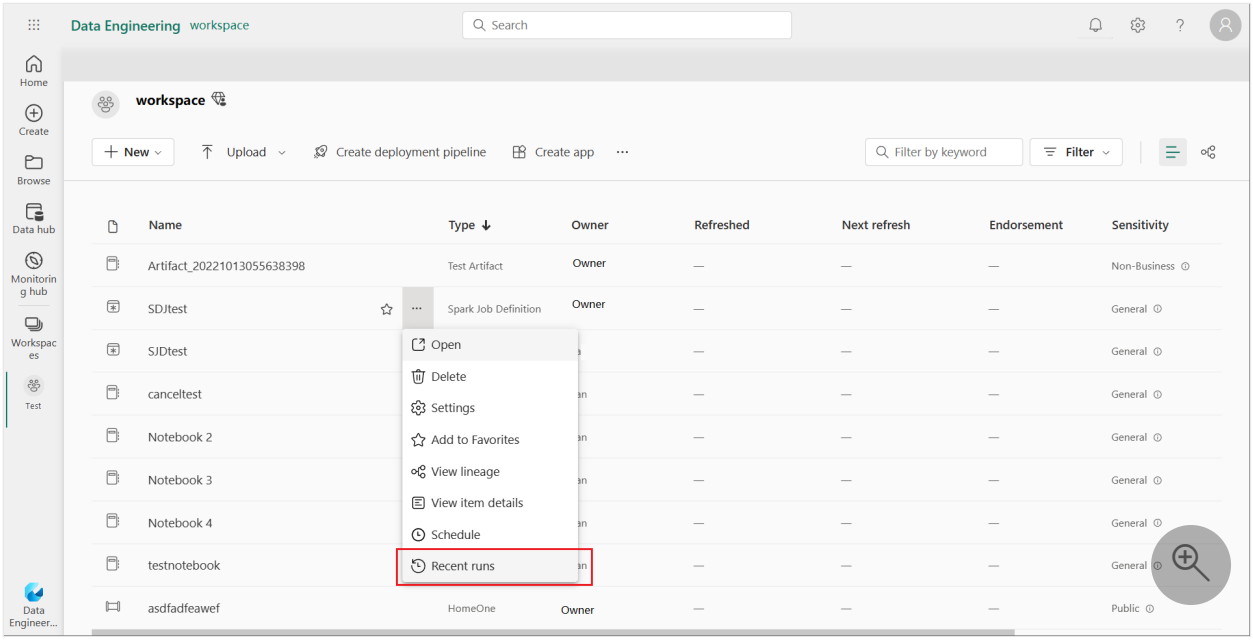
Spark job definition inline monitoring

The Spark job definition inline monitoring feature allows you to view Spark job definition submission and run status in real-time. You can also view the Spark job definition's past runs and configurations and navigate to the **Spark application detail** page to view more details.



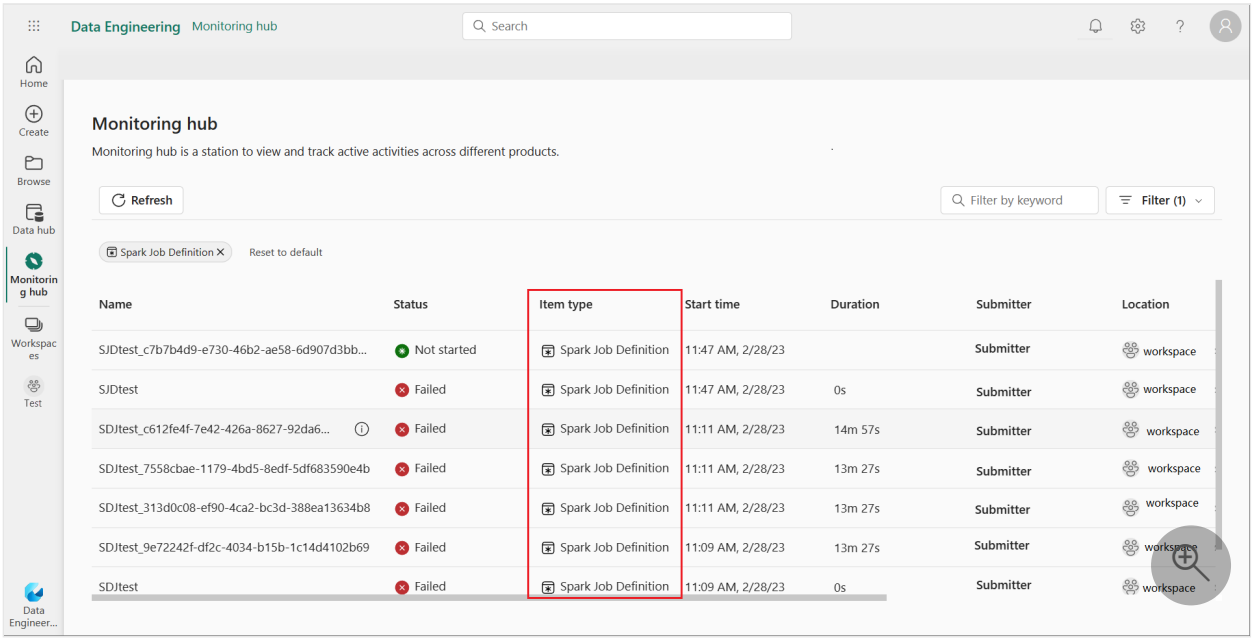
Spark job definition item view in workspace

You can access the job runs associated with specific Spark job definition items by using the **Recent run** contextual menu on the workspace homepage.



Spark job definition runs in the Monitoring hub

To view all the Spark applications related to a Spark job definition, go to the **Monitoring hub**. Sort or filter the **Item Type** column to view all the run activities associated with the Spark job definitions.



Next steps

The next step after viewing the details of an Apache Spark application is to view Spark job progress below the Notebook cell. You can refer to

- Spark application detail monitoring

Apache Spark application detail monitoring

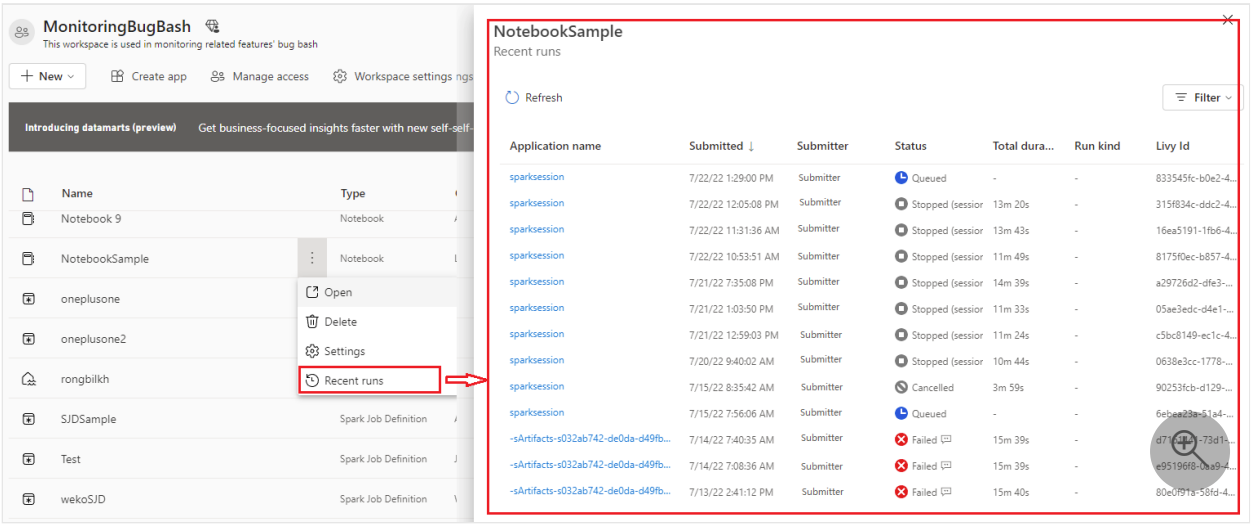
Article • 05/23/2023

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

With Microsoft Fabric, you can use Apache Spark to run notebooks, jobs, and other kinds of applications in your workspace. This article explains how to monitor your Apache Spark application, allowing you to keep an eye on the recent run status, issues, and progress of your jobs.

View Apache Spark applications

You can view all Apache Spark applications from **Spark job definition**, or **notebook item context** menu shows the recent run option -> **Recent runs**.



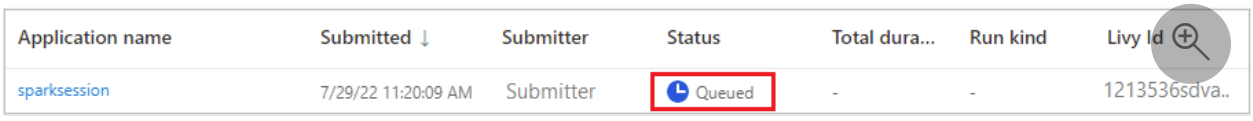
You can select the name of the application you want to view in the application list, in the application details page you can view the application details.

Monitor Apache Spark application status

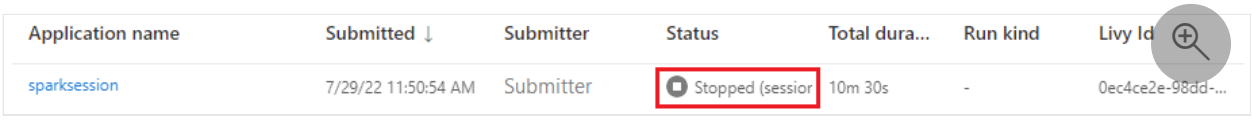
Open the **Recent runs** page of the notebook or Spark job definition, you can view the status of the Apache application.

- Success

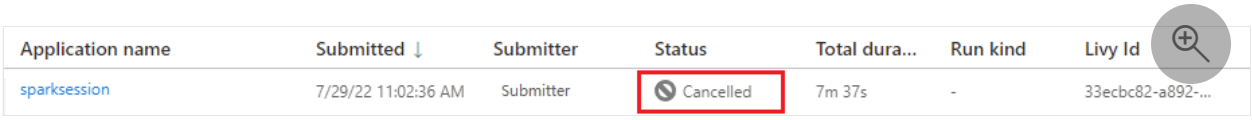
- Queued



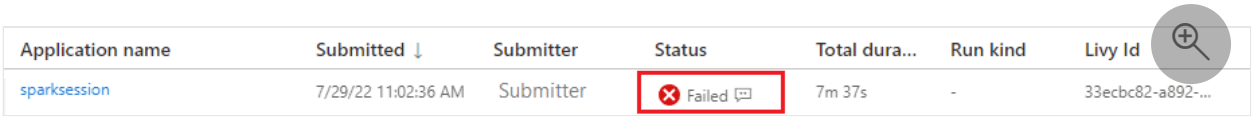
- Stopped



- Canceled



- Failed

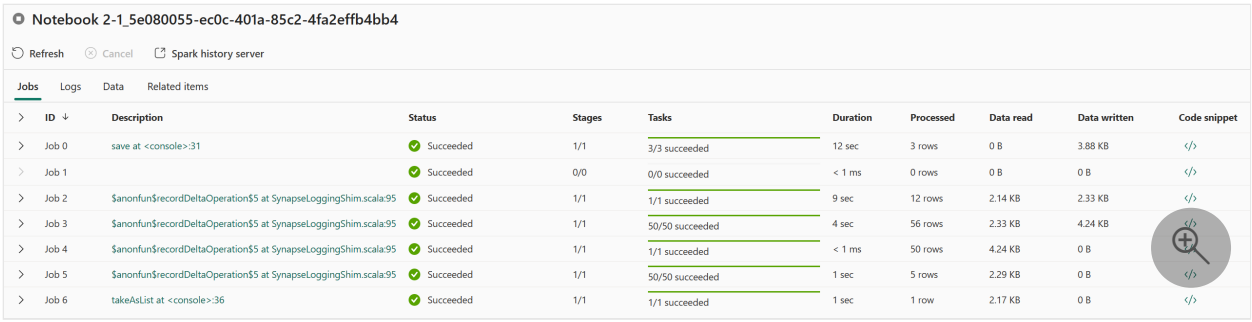


Jobs

Open an Apache Spark application job from the **Spark job definition** or **notebook** item context menu shows the **Recent run** option -> **Recent runs** -> select a job in recent runs page.

In the Apache Spark application monitoring details page, the job runs list is displayed in the **Jobs** tab, you can view the details of each job here, including **Job ID**, **Description**, **Status**, **Stages**, **Tasks**, **Duration**, **Processed**, **Data read**, **Data written** and **Code snippet**.

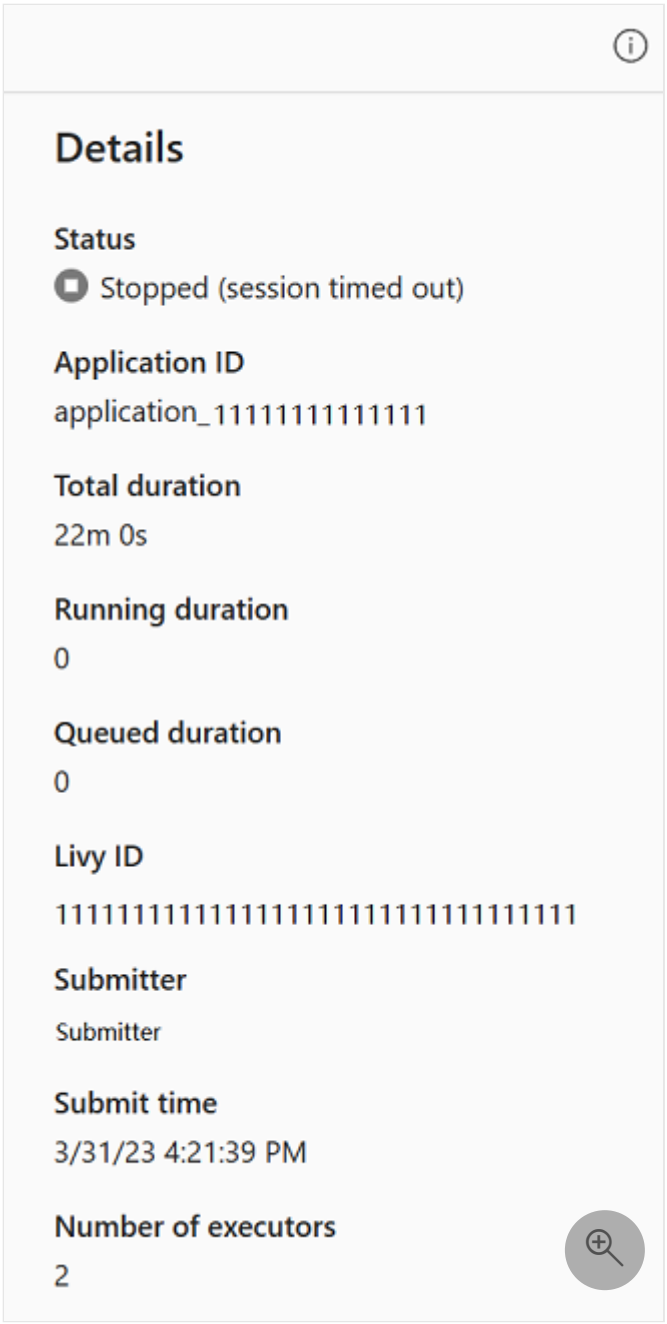
- Clicking on Job ID can expand/collapse the job.
- Click on the job description, you can jump to job or stage page in spark UI.
- Click on the job Code snippet, you can check and copy the code related to this job.



Summary panel

In the Apache Spark application monitoring page, click the **Properties** button to open/collapse the summary panel. You can view the details for this application in **Details**.

- Status for this spark application.
- This Spark application's ID.
- Total duration.
- Running duration for this spark application.
- Queued duration for this spark application.
- Livy ID
- Submitter for this spark application.
- Submit time for this spark application.
- Number of executors.

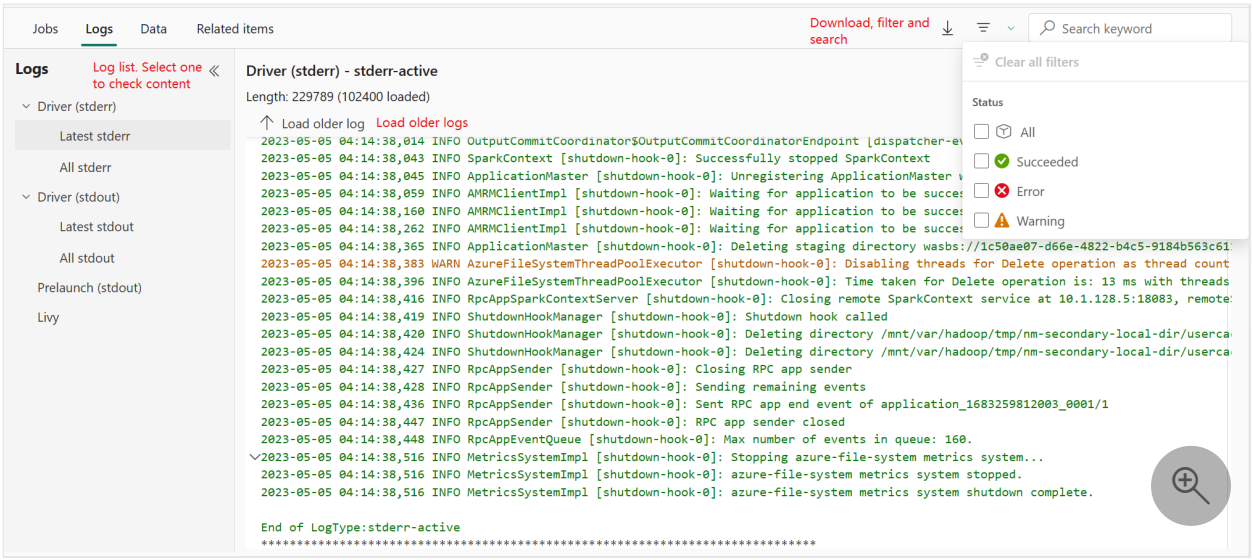


Logs

For the **Logs** tab, you can view the full log of **Livy**, **Prelaunch**, **Driver** log with different options selected in the left panel. And you can directly retrieve the required log information by searching keywords and view the logs by filtering the log status. Click Download Log to download the log information to the local.

Sometimes no logs are available, such as the status of the job is queued and cluster creation failed.

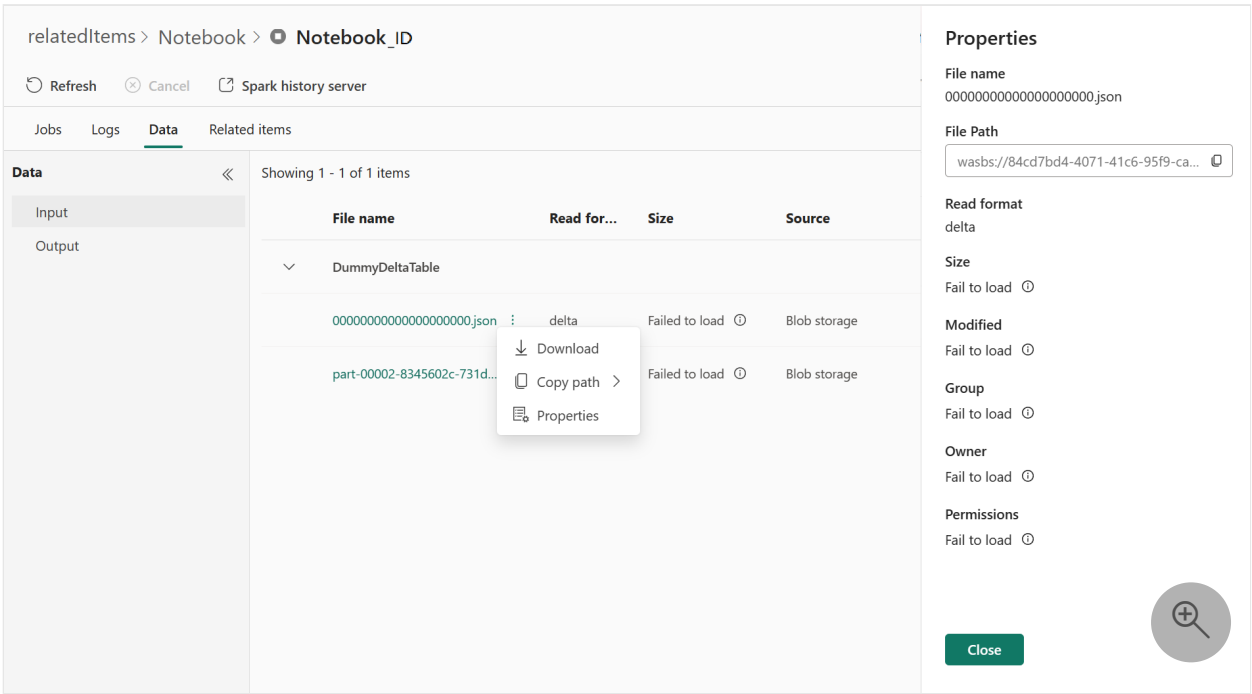
Live logs are only available when app submission fails, and driver logs are also provided.



Data

For the **Data** tab, you can copy the data list on clipboard, download the data list and single data, and check the properties for each data.

- The left panel can be expanded or collapse.
- The name, read format, size, source and path of the input and output files will be displayed in this list.
- The files in input and output can be downloaded, copy path and view properties.



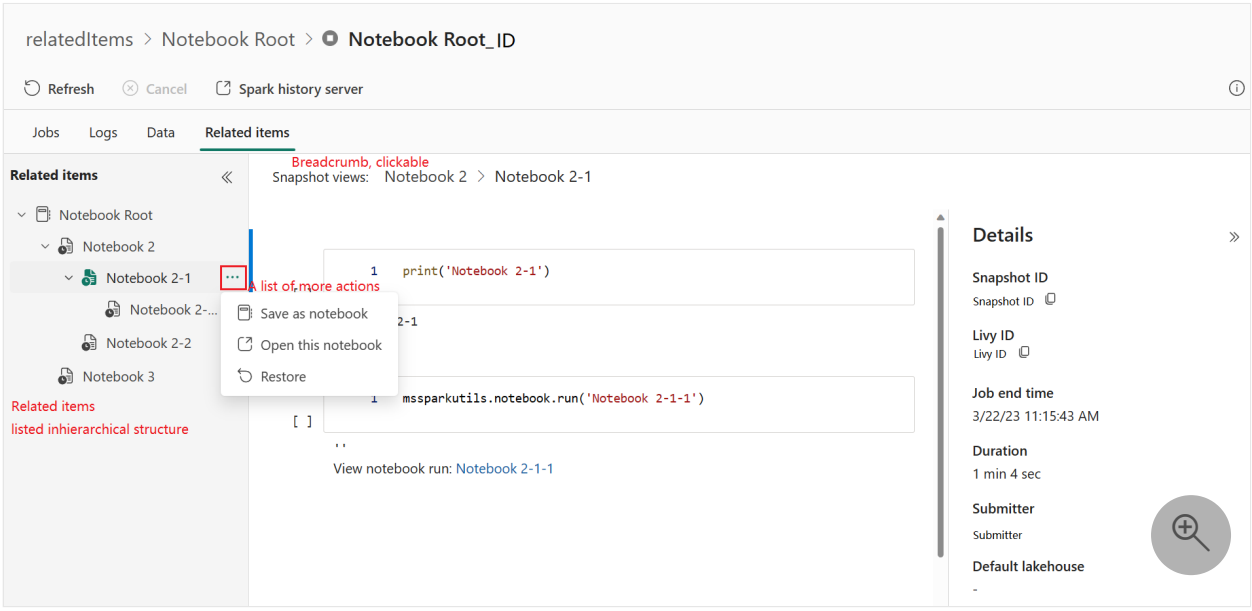
Related items

The **Related items** tab allows you to browse and view items associated with the Apache Spark application, including Notebooks, Spark job definition, and/or Pipelines. The

related items page displays the snapshot of the code and parameter values at the time of execution for Notebooks. It also shows the snapshot of all settings and parameters at the time of submission for Spark job definitions. If the Apache Spark application is associated with a pipeline, the related item page also presents the corresponding pipeline and the Spark activity.

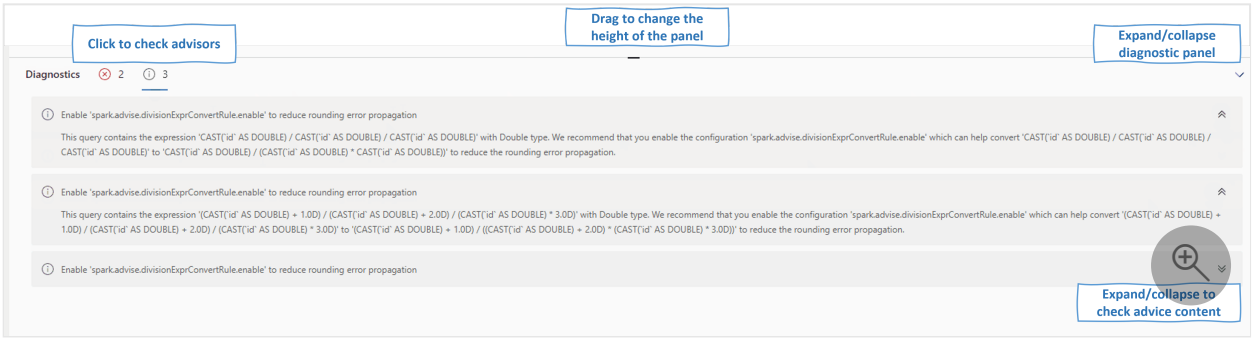
In the Related Items screen, you can:

- Browse and navigate the related items in the hierarchical tree.
- Click the 'A list of more actions' ellipse icon for each item to take different actions.
- Click the snapshot item to view its content.
- View the Breadcrumb to see the path from the selected item to the root.



Diagnostics

The diagnostic panel provides users with real-time recommendations and error analysis, which are generated by the Spark Advisor through an analysis of the user's code. With built-in patterns, the Apache Spark Advisor helps users avoid common mistakes and analyzes failures to identify their root cause.



Next steps

The next step after viewing the details of an Apache Spark application is to view **Spark job progress** below the Notebook cell. You can refer to:

- Notebook contextual monitoring and debugging

Use extended Apache Spark history server to debug and diagnose Apache Spark applications

Article • 05/23/2023

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

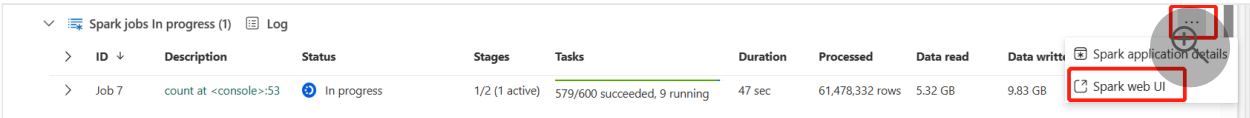
This article provides guidance on how to use the extended Apache Spark history server to debug and diagnose completed and running Spark applications.

Access the Apache Spark history server

Apache Spark history server is the web user interface for completed and running Spark applications. You can open the Apache Spark web UI interface from the progress indicator notebook or Apache Spark application detail page.

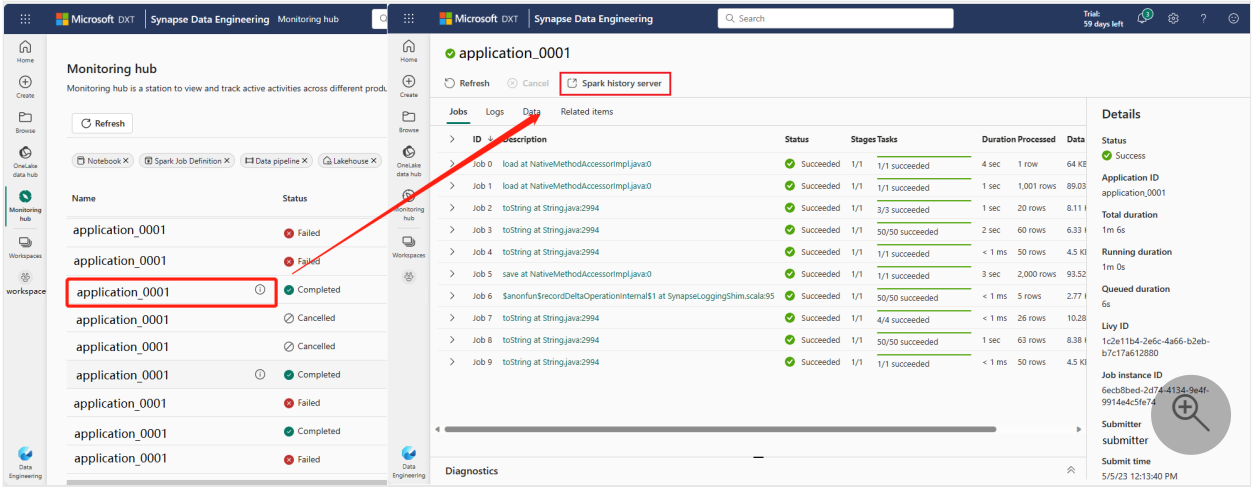
Open the Spark web UI from progress indicator notebook

When an Apache Spark job is triggered, the button to open **Spark web UI** will be inside the **More action** in the progress indicator. Click on **Spark web UI** and wait for a few seconds, then the Spark UI page will show up.

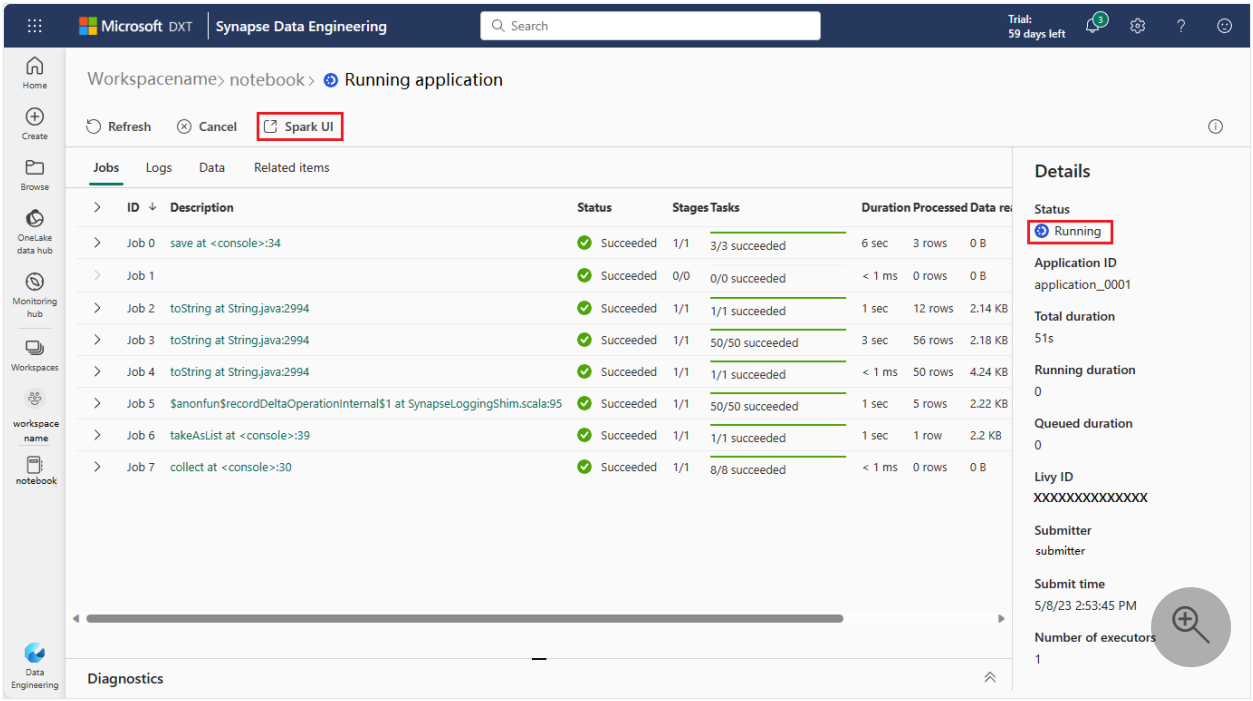


Open the Spark web UI from Apache Spark application detail page

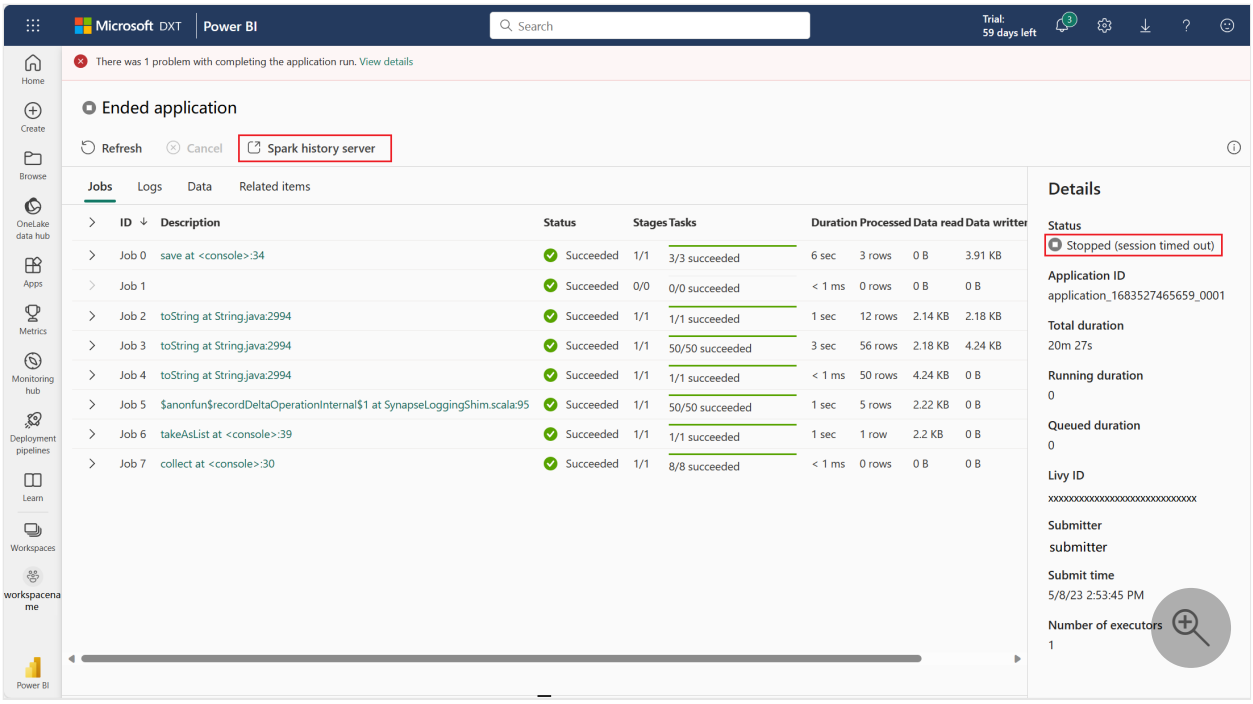
Spark web UI can also be opened through Apache Spark application detail page. Select **Monitoring hub** on the left, and then select an Apache Spark application. Then the page will be redirected to the detail page of the application.



For Apache Spark application whose status is running, the button shows **Spark UI**. Click on **Spark UI**, the Spark UI page will show up.



For an Apache Spark application whose status is ended, the status of ended includes **Stopped**, **Failed**, **Canceled**, and **Completed**. The button shows **Spark history server**. Click on **Spark history server**, the Spark UI page will show up.

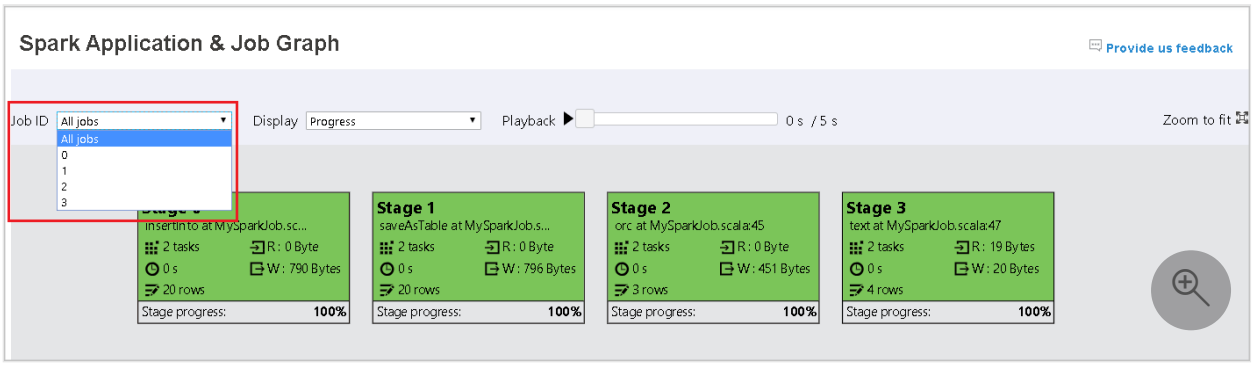


Graph tab in Apache Spark history server

Select the Job ID for the job you want to view. Then, select **Graph** on the tool menu to get the job graph view.

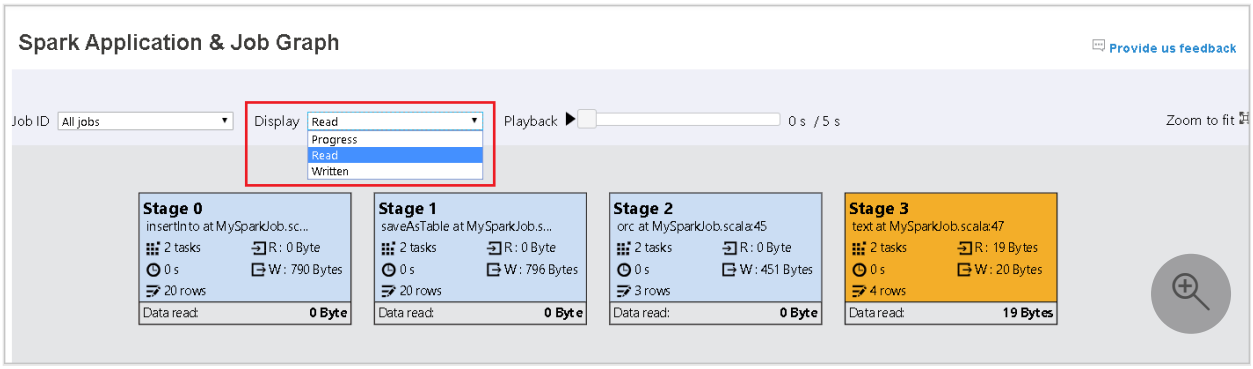
Overview

You can see an overview of your job in the generated job graph. By default, the graph shows all jobs. You can filter this view by **Job ID**.

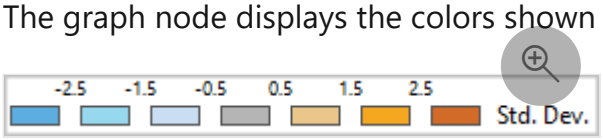


Display

By default, the **Progress** display is selected. You can check the data flow by selecting **Read** or **Written** in the **Display** dropdown list.



The graph node displays the colors shown in the heatmap legend.

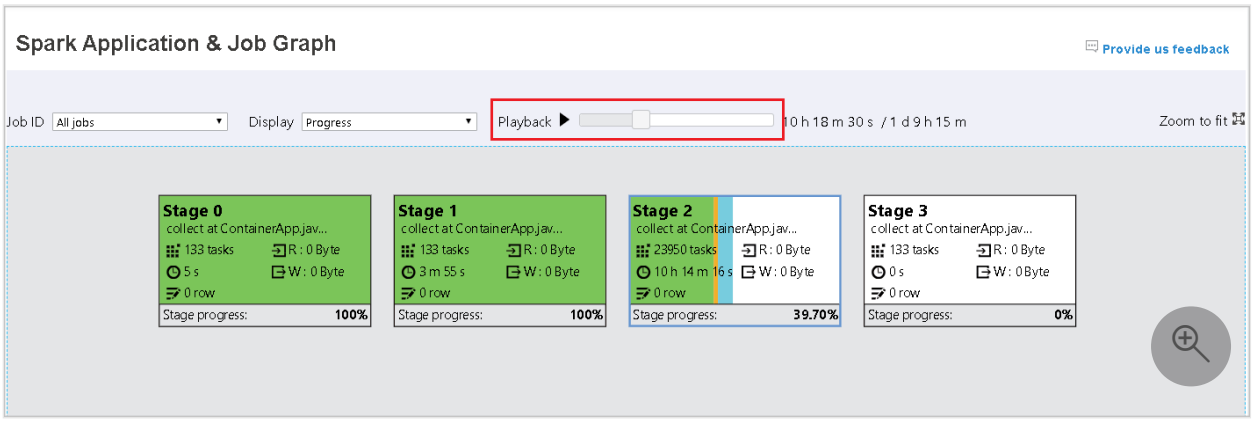


Playback

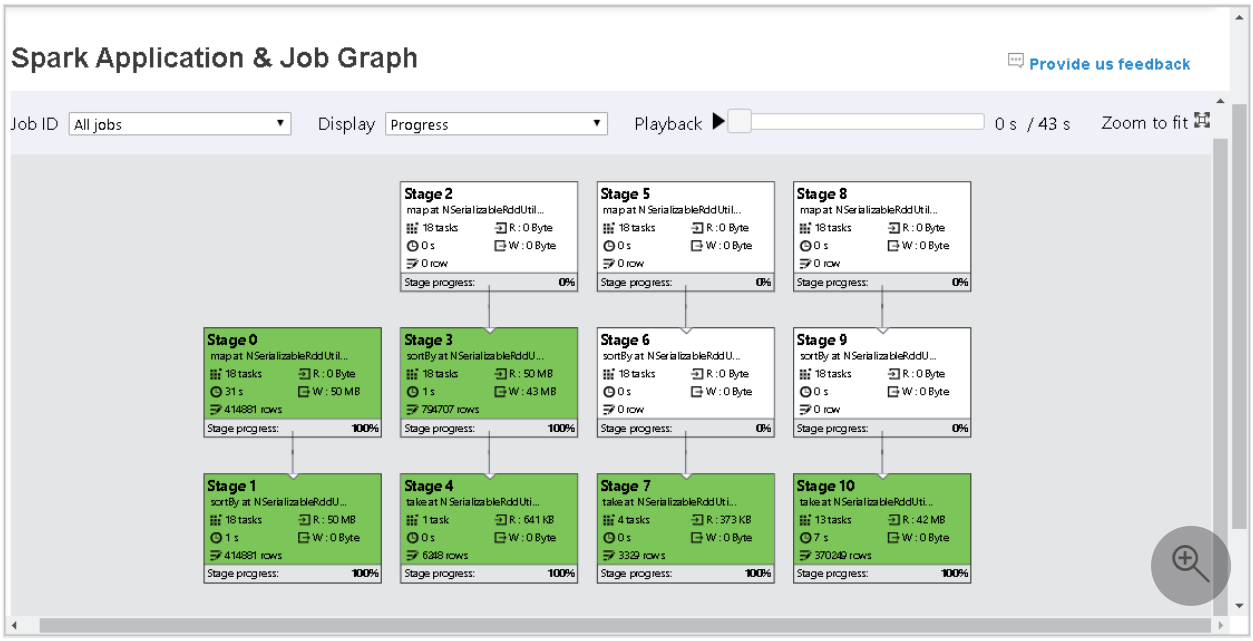
To playback the job, select **Playback**. You can select **Stop** at any time to stop. The task colors show different statuses when playing back:

Color	Meaning
Green	Succeeded: The job has completed successfully.
Orange	Retried: Instances of tasks that failed but do not affect the final result of the job. These tasks had duplicate or retry instances that may succeed later.
Blue	Running: The task is running.
White	Waiting or skipped: The task is waiting to run, or the stage has skipped.
Red	Failed: The task has failed.

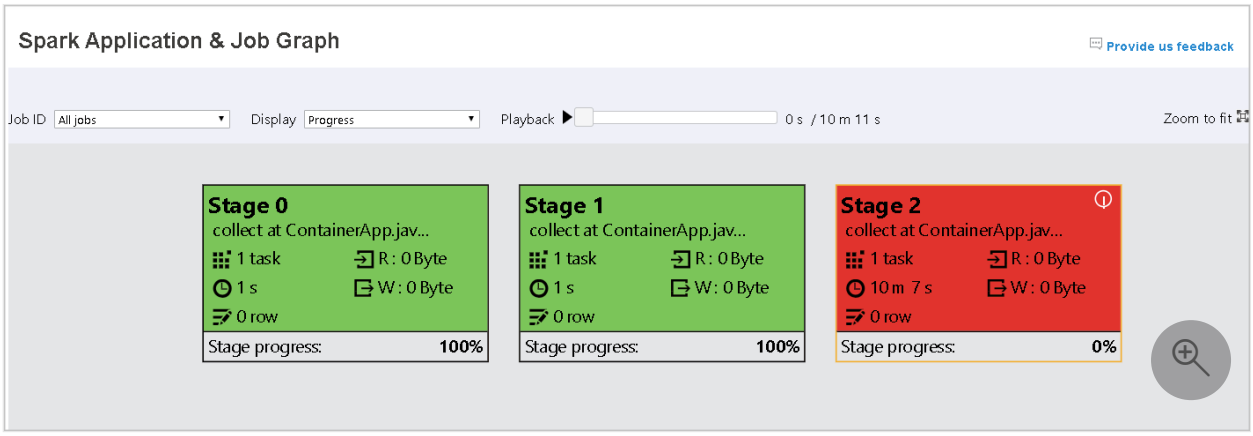
The following image shows Green, Orange, and Blue status colors.



The following image shows Green and White status colors.



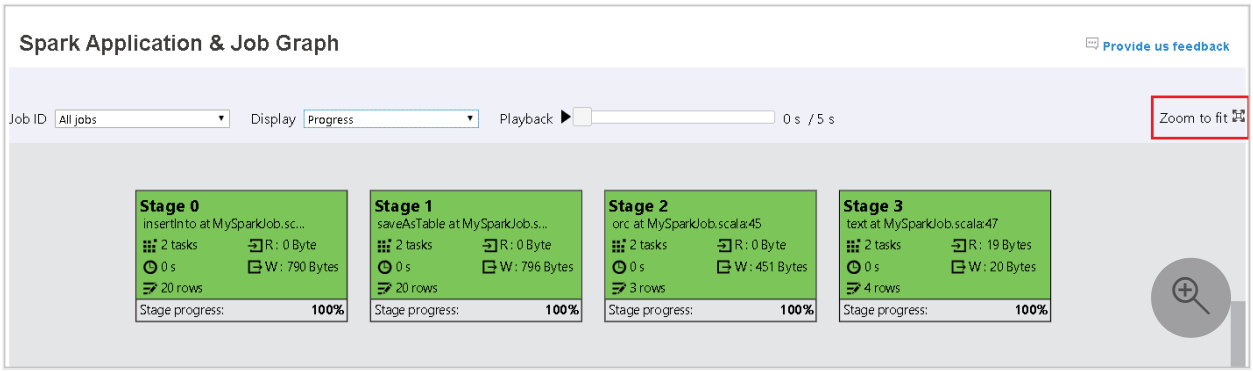
The following image shows Red and Green status colors.



> **Note**
>
> Playback for each job is allowed. For incomplete jobs, playback is not supported.

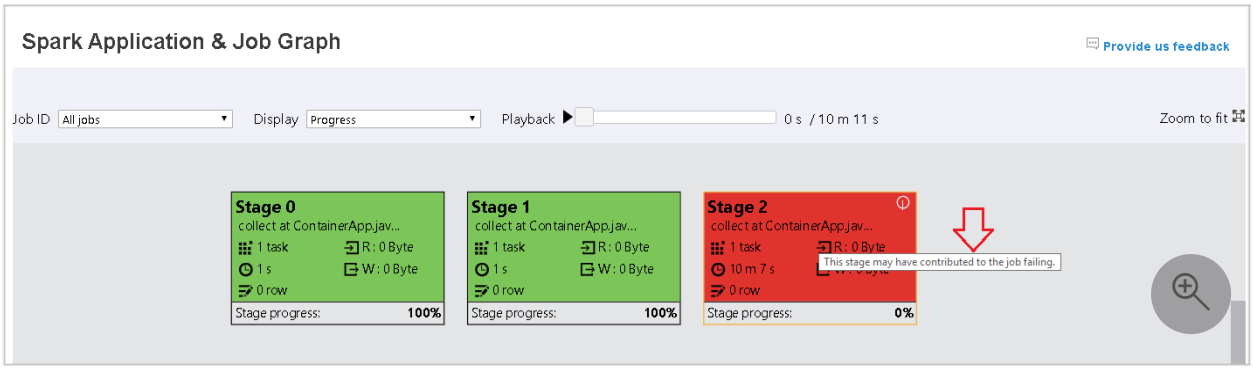
Zoom

Use your mouse scroll to zoom in and out on the job graph, or select **Zoom to fit** to make it fit to screen.



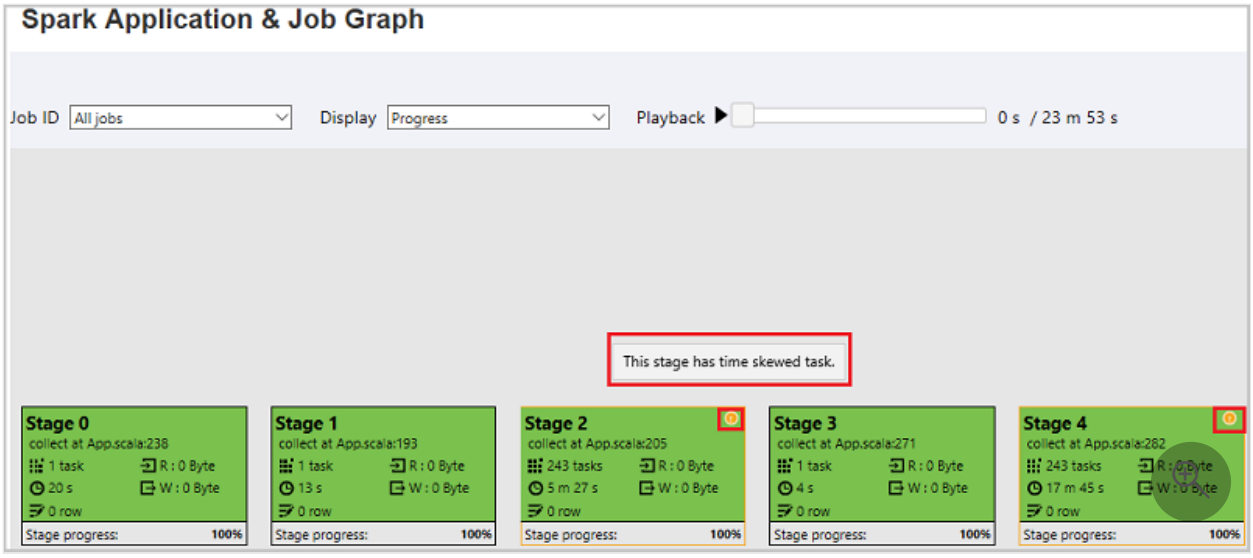
Tooltips

Hover on graph node to see the tooltip when there are failed tasks, and select a stage to open its stage page.



On the job graph tab, stages have a tooltip and a small icon displayed if they have tasks that meet the following conditions:

Condition	Description
Data skew	data read size > average data read size of all tasks inside this stage * 2 and data read size > 10 MB
Time skew	execution time > average execution time of all tasks inside this stage * 2 and execution time > 2 minutes



Graph node description

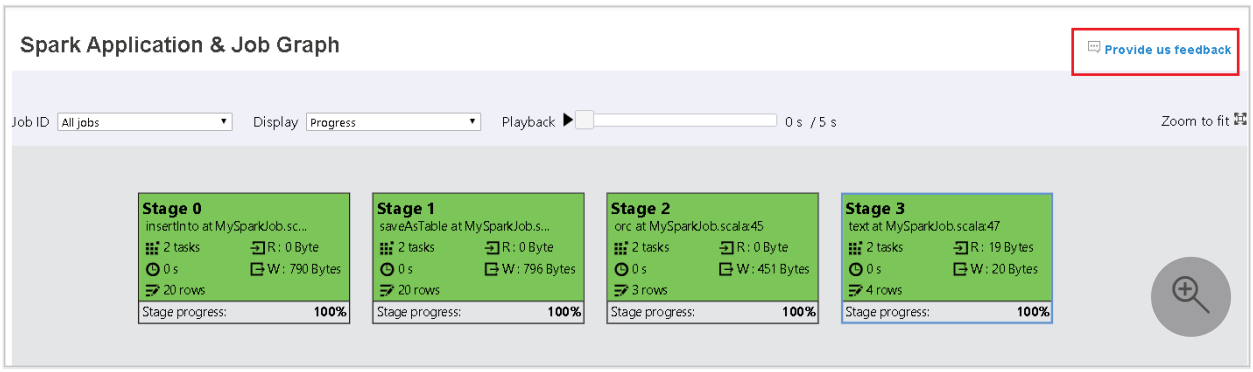
The job graph node displays the following information of each stage:

- ID.
- Name or description.
- Total task number.
- Data read: the sum of input size and shuffle read size.
- Data write: the sum of output size and shuffle writes size.
- Execution time: the time between start time of the first attempt and completion time of the last attempt.
- Row count: the sum of input records, output records, shuffle read records and shuffle write records.
- Progress.

> **Note**
>
> By default, the job graph node displays information from the last attempt of each stage (except for stage execution time). However, during playback, the graph node shows information of each attempt.
>
> The data size of read and write is 1MB = 1000 KB = 1000 * 1000 Bytes.

Provide feedback

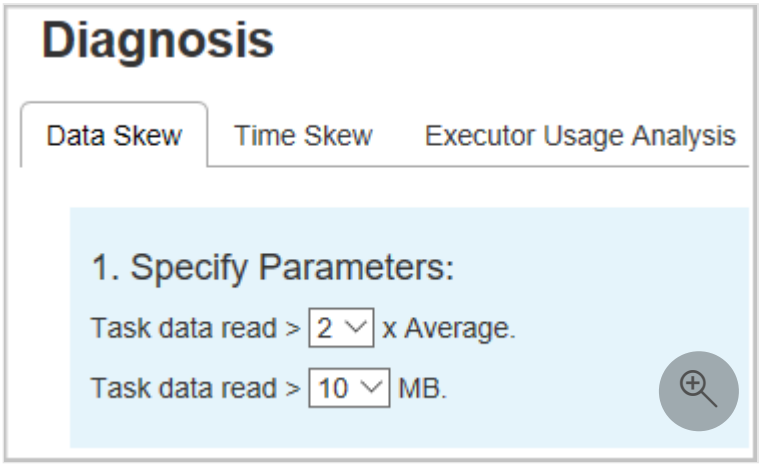
Send feedback with issues by selecting **Provide us feedback**.



Explore the Diagnosis tab in Apache Spark history server

To access the Diagnosis tab, select a job ID. Then select **Diagnosis** on the tool menu to get the job Diagnosis view. The diagnosis tab includes **Data Skew**, **Time Skew**, and **Executor Usage Analysis**.

Check the **Data Skew**, **Time Skew**, and **Executor Usage Analysis** by selecting the tabs respectively.

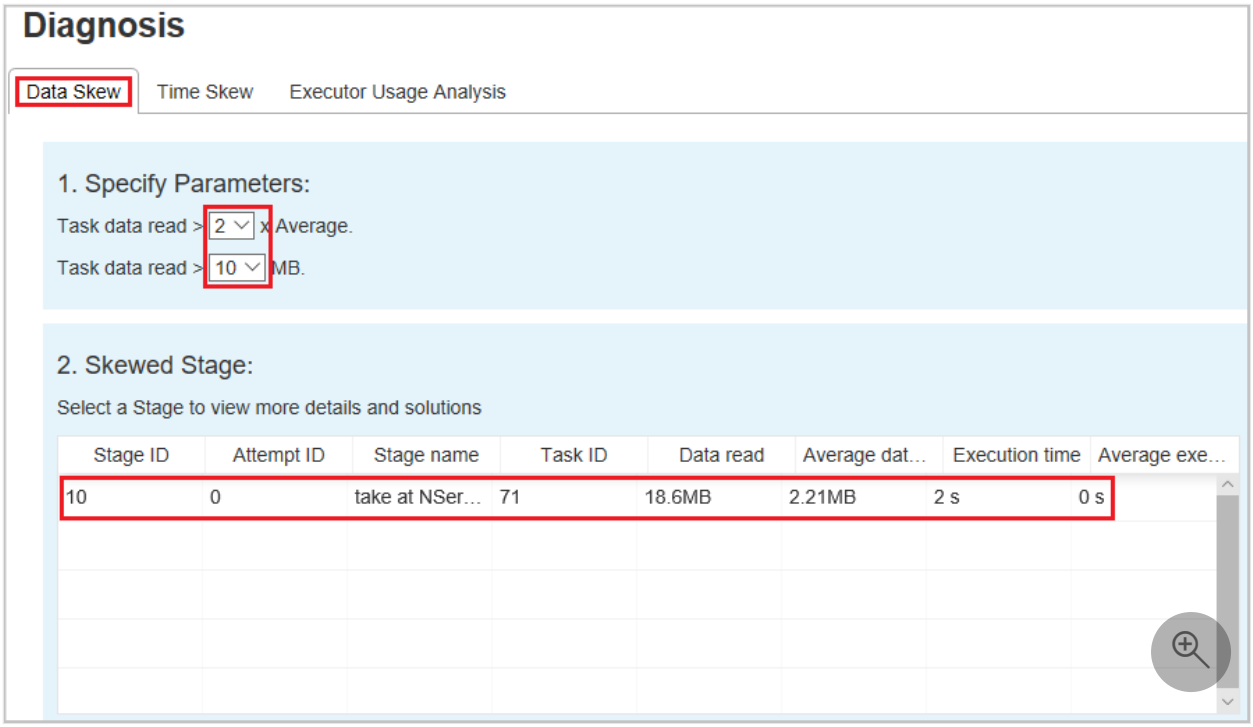


Data Skew

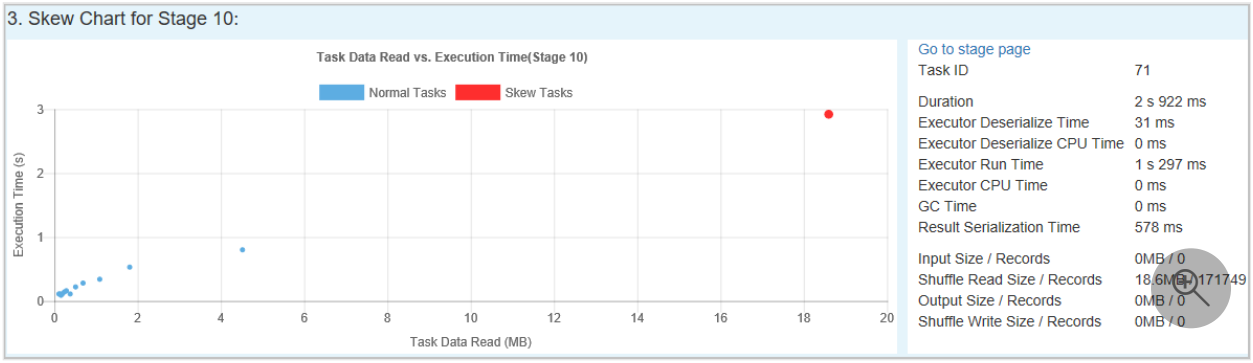
When you select the **Data Skew** tab, the corresponding skewed tasks are displayed based on the specified parameters.

- **Specify Parameters** - The first section displays the parameters, which are used to detect Data Skew. The default rule is: Task Data Read is greater than three times of the average task data read, and the task data read is more than 10 MB. If you want to define your own rule for skewed tasks, you can choose your parameters. The **Skewed Stage** and **Skew Char** sections are refreshed accordingly.

- **Skewed Stage** - The second section displays stages, which have skewed tasks meeting the criteria specified above. If there is more than one skewed task in a stage, the skewed stage table only displays the most skewed task (for example, the largest data for data skew).



- **Skew Chart** – When a row in the skew stage table is selected, the skew chart displays more task distribution details based on data read and execution time. The skewed tasks are marked in red and the normal tasks are marked in blue. The chart displays up to 100 sample tasks, and the task details are displayed in right bottom panel.



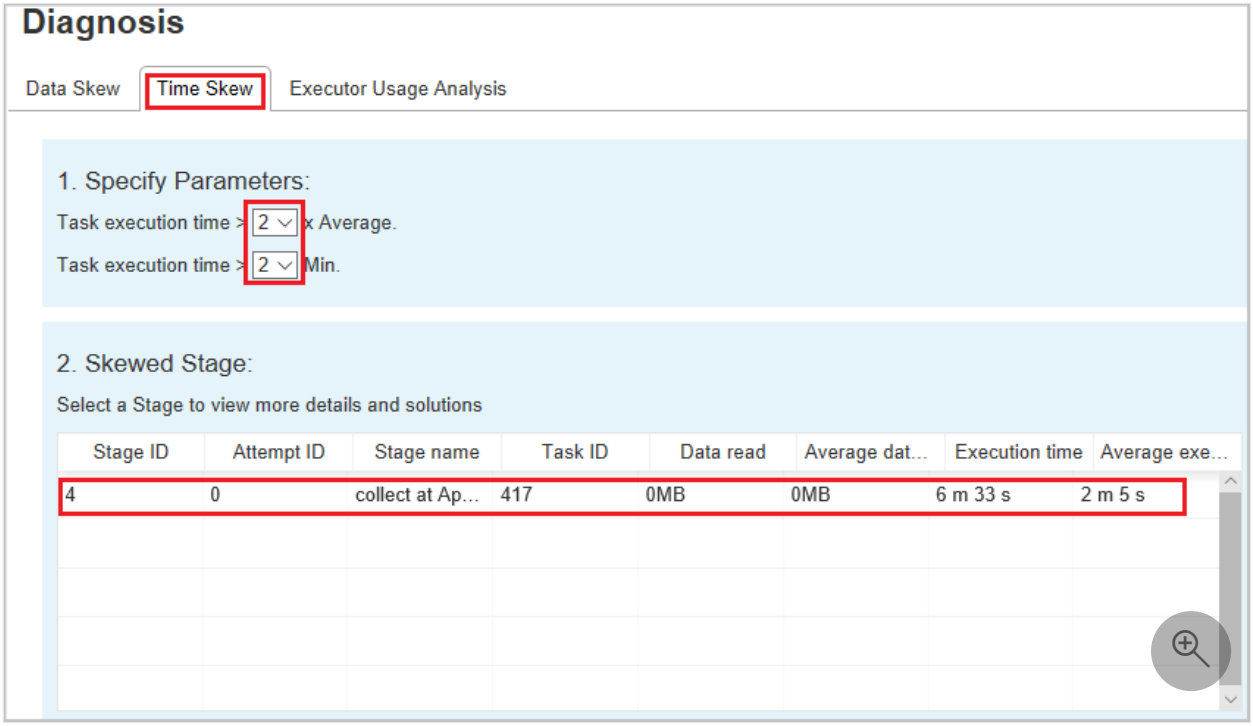
Time Skew

The **Time Skew** tab displays skewed tasks based on task execution time.

- **Specify Parameters** - The first section displays the parameters, which are used to detect Time Skew. The default criteria to detect time skew is: task execution time is greater than three times of average execution time and task execution time is

greater than 30 seconds. You can change the parameters based on your needs. The **Skewed Stage** and **Skew Chart** display the corresponding stages and tasks information just like the **Data Skew** tab above.

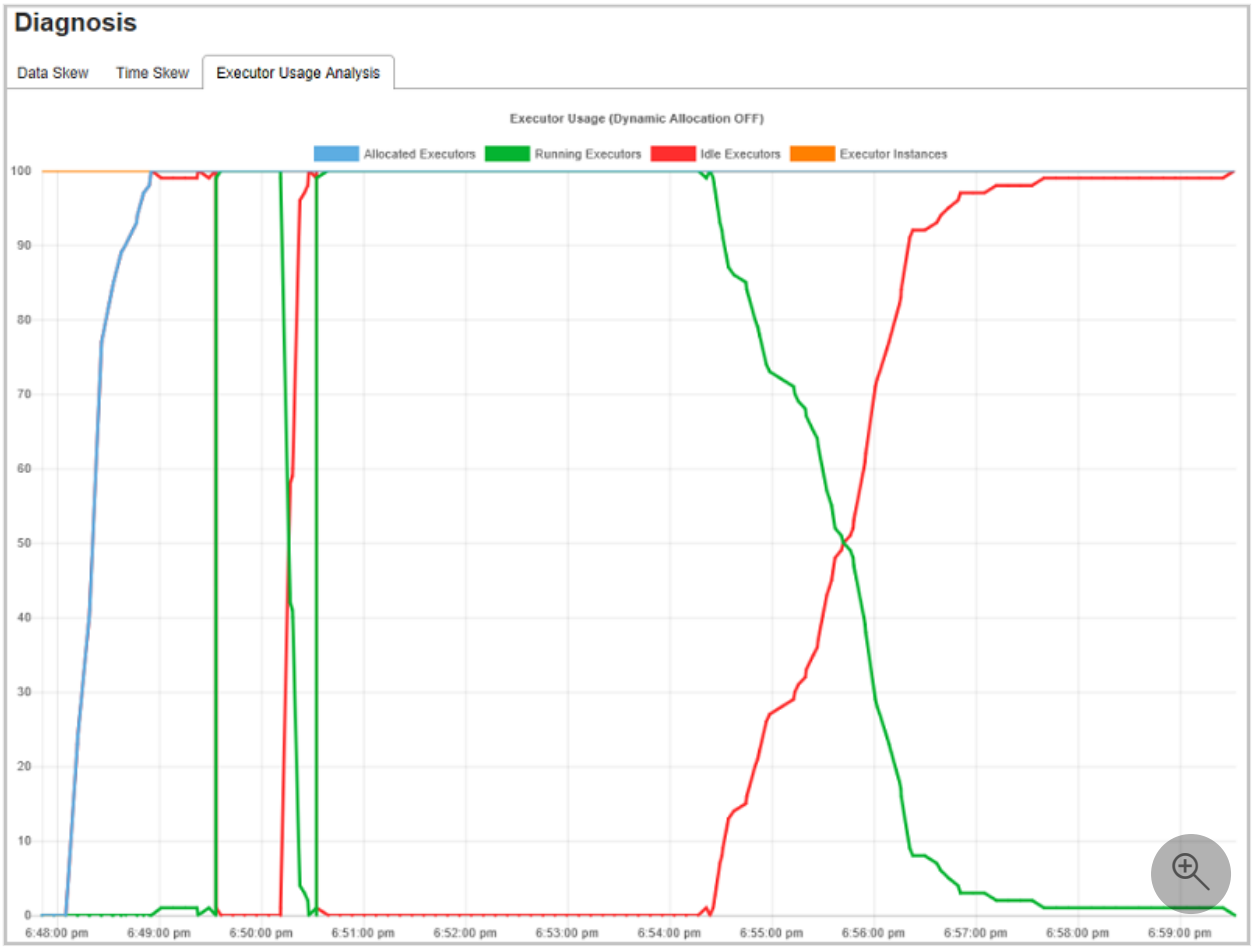
- Select **Time Skew**, then filtered result is displayed in **Skewed Stage** section according to the parameters set in section **Specify Parameters**. Select one item in **Skewed Stage** section, then the corresponding chart is drafted in section 3, and the task details are displayed in right bottom panel.



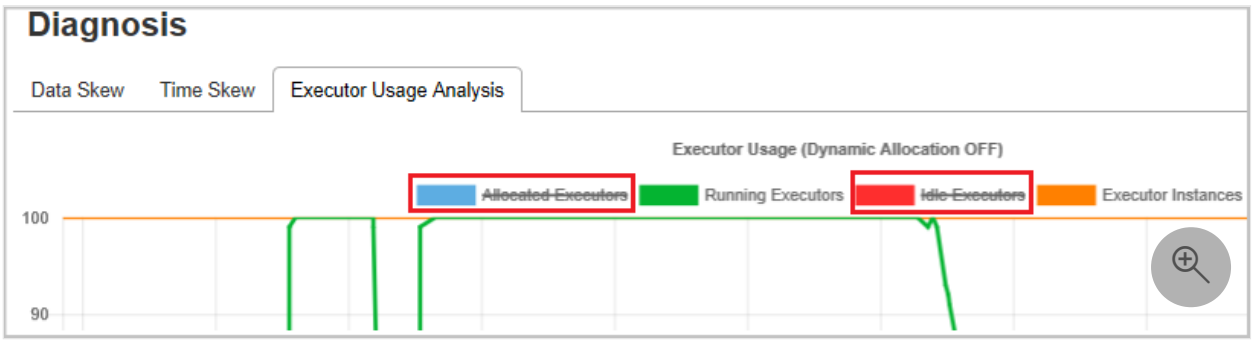
Executor Usage Analysis

The Executor Usage Graph visualizes the Spark job executor's allocation and running status.

1. Select **Executor Usage Analysis**, then four types curves about executor usage are drafted, including **Allocated Executors**, **Running Executors**, **Idle Executors**, and **Max Executor Instances**. For allocated executors, each "Executor added" or "Executor removed" event increases or decreases the allocated executors. You can check "Event Timeline" in the "Jobs" tab for more comparison.



2. Select the color icon to select or unselect the corresponding content in all drafts.



Next steps

- Apache Spark monitoring overview
- Browse item's recent runs
- Monitor Apache Spark jobs within notebooks
- Monitor Apache Spark job definition
- Monitor Apache Spark application details

Monitor Spark capacity consumption

Article • 05/23/2023

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

The purpose of this article is to offer guidance for admins who want to monitor activities in the capacities they manage. By utilizing the Spark capacity consumption reports available in the Microsoft Fabric utilization and metrics app, admins can gain insights into the billable Spark capacity consumption for Spark workload items, including Lakehouse, Notebook, and Spark job definitions. Some Spark capacity consumption activities aren't reported in the app.

Spark capacity consumption reported

The following operations from Lakehouse, Notebook, and Spark job definitions will be treated as billable activities once we move out of the preview status. During public preview, the capacity consumption from these operations will be displayed as *Preview* in the metrics app.

Operation name	Item	Timeline	Comments
Lakehouse operations	Lakehouse	Public preview	Users preview table in the Lakehouse explorer.
Lakehouse table load	Lakehouse	Public preview	Users load delta table in the Lakehouse explorer.
Notebook run	Synapse Notebook	Public preview	Synapse Notebook runs manually by users.
Notebook HC run	Synapse Notebook	Public preview	Synapse Notebook runs under the high concurrency Spark session.
Notebook scheduled run	Synapse Notebook	Public preview	Synapse Notebook runs triggered by notebook scheduled events.
Notebook pipeline run	Synapse Notebook	Public preview	Synapse Notebook runs triggered by pipeline.

Operation name	Item	Timeline	Comments
Notebook VS Code run	Synapse Notebook	Public preview	Synapse Notebook runs in VS Code.
Spark job run	Spark Job Definition	Public preview	Spark batch job runs initiated by user submission.
Spark job scheduled run	Spark Job Definition	Public preview	Synapse batch job runs triggered by notebook scheduled events.
Spark job pipeline run	Spark Job Definition	Public preview	Synapse batch job runs triggered by pipeline.
Spark job VS Code run	Spark Job Definition	Public preview	Synapse Spark job definition submitted from VS Code.

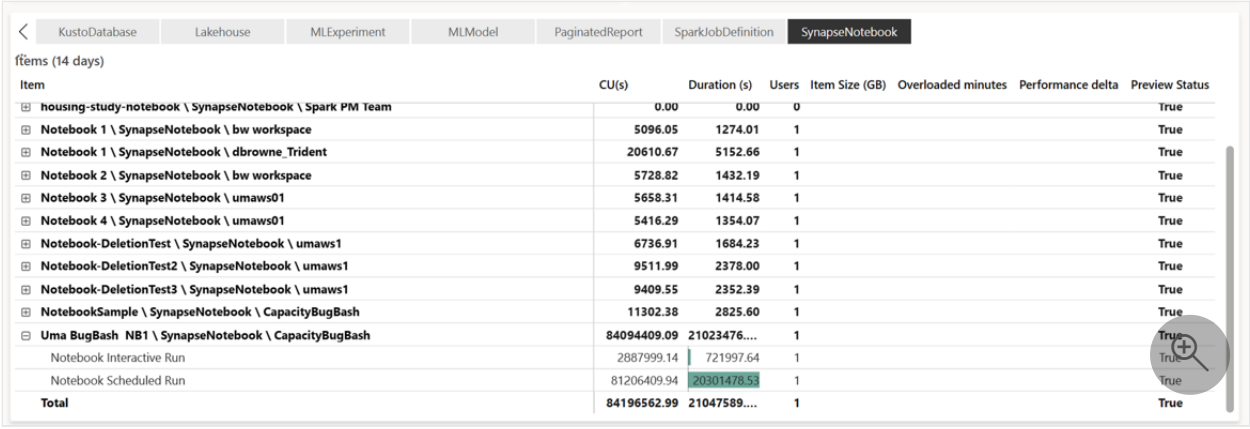
Spark capacity consumption that isn’t reported

There are some Spark capacity consumption activities that aren't reported in the metrics app. These activities include system Spark jobs for Library Management and certain system Spark jobs for Spark Live pool or live sessions.

- **Library Management** - The capacity consumption associated with Library Management at the workspace level and environment level Library Management, isn't reported in the metrics app.
- **System Spark jobs** - Spark capacity consumption that isn't associated with a Notebook, a Spark Job Definition, or a Lakehouse, isn't included in the capacity reporting.

Capacity consumption reports

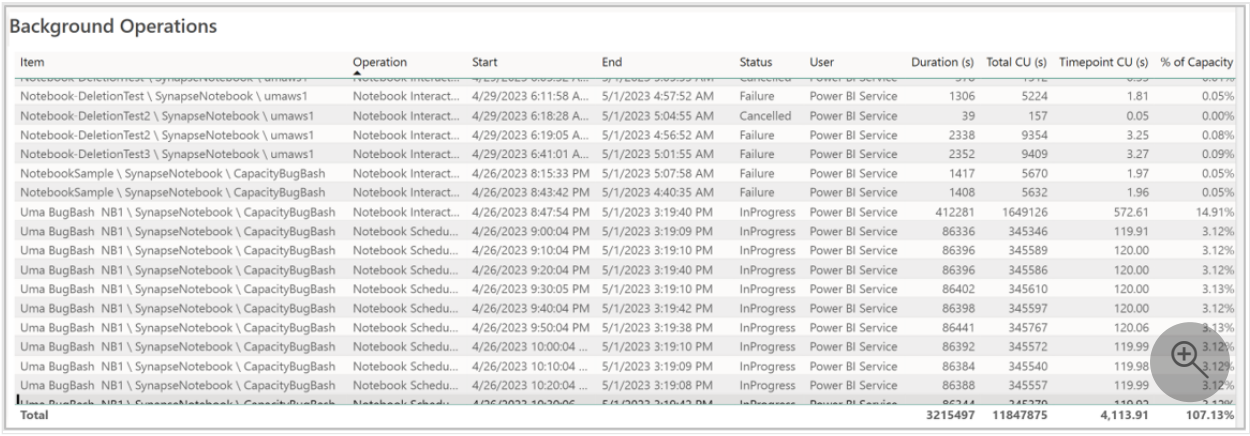
All Spark related operations are classified as background operations. Capacity consumption from Spark is displayed under a Notebook, a Spark Job Definition, or a Lakehouse, and is aggregated by operation name and item.



Item	CU(s)	Duration (s)	Users	Item Size (GB)	Overloaded minutes	Performance delta	Preview Status
housing-study-notebook \ SynapseNotebook \ Spark PM Team	0.00	0.00	0				True
Notebook 1 \ SynapseNotebook \ bw workspace	5096.05	1274.01	1				True
Notebook 1 \ SynapseNotebook \ dbrowne_Trident	20610.67	5152.66	1				True
Notebook 2 \ SynapseNotebook \ bw workspace	5728.82	1432.19	1				True
Notebook 3 \ SynapseNotebook \ umaws01	5658.31	1414.58	1				True
Notebook 4 \ SynapseNotebook \ umaws01	5416.29	1354.07	1				True
Notebook-DeletionTest \ SynapseNotebook \ umaws1	6736.91	1684.23	1				True
Notebook-DeletionTest2 \ SynapseNotebook \ umaws1	9511.99	2378.00	1				True
Notebook-DeletionTest3 \ SynapseNotebook \ umaws1	9409.55	2352.39	1				True
NotebookSample \ SynapseNotebook \ CapacityBugBash	11302.38	2825.60	1				True
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	84094409.09	21023476....	1				True
Notebook Interactive Run	2887999.14	721997.64	1				True
Notebook Scheduled Run	81206409.94	20301478.53	1				True
Total	84196562.99	21047589....	1				True

Background Operations report

Background operations are displayed for a specific timepoint. In the report's table, each row refers to a user operation. Review the *User* column to identify who performed a specific operation. If you need more information about a specific operation, you can use its *Operation ID* to look it up in the Microsoft Fabric monitoring hub.



Item	Operation	Start	End	Status	User	Duration (s)	Total CU (s)	Timepoint CU (s)	% of Capacity
Notebook-DeletionTest \ SynapseNotebook \ umaws1	Notebook Interact...	4/29/2023 6:11:58 A...	5/1/2023 4:57:52 AM	Failure	Power BI Service	1306	5224	1.81	0.05%
Notebook-DeletionTest2 \ SynapseNotebook \ umaws1	Notebook Interact...	4/29/2023 6:18:28 A...	5/1/2023 5:04:55 AM	Cancelled	Power BI Service	39	157	0.05	0.00%
Notebook-DeletionTest2 \ SynapseNotebook \ umaws1	Notebook Interact...	4/29/2023 6:19:05 A...	5/1/2023 4:56:52 AM	Failure	Power BI Service	2338	9354	3.25	0.08%
Notebook-DeletionTest3 \ SynapseNotebook \ umaws1	Notebook Interact...	4/29/2023 6:41:01 A...	5/1/2023 5:01:55 AM	Failure	Power BI Service	2352	9409	3.27	0.09%
NotebookSample \ SynapseNotebook \ CapacityBugBash	Notebook Interact...	4/26/2023 8:15:33 PM	5/1/2023 5:07:58 AM	Failure	Power BI Service	1417	5670	1.97	0.05%
NotebookSample \ SynapseNotebook \ CapacityBugBash	Notebook Interact...	4/26/2023 8:43:42 PM	5/1/2023 4:40:35 AM	Failure	Power BI Service	1408	5632	1.96	0.05%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Interact...	4/26/2023 8:47:54 PM	5/1/2023 3:19:40 PM	InProgress	Power BI Service	412281	1649126	572.61	14.91%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 9:00:04 PM	5/1/2023 3:19:09 PM	InProgress	Power BI Service	86336	345346	119.91	3.12%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 9:10:04 PM	5/1/2023 3:19:10 PM	InProgress	Power BI Service	86396	345589	120.00	3.12%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 9:20:04 PM	5/1/2023 3:19:40 PM	InProgress	Power BI Service	86396	345586	120.00	3.12%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 9:30:05 PM	5/1/2023 3:19:10 PM	InProgress	Power BI Service	86402	345610	120.00	3.13%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 9:40:04 PM	5/1/2023 3:19:42 PM	InProgress	Power BI Service	86398	345597	120.00	3.12%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 9:50:04 PM	5/1/2023 3:19:38 PM	InProgress	Power BI Service	86441	345767	120.06	3.13%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 10:00:04 ...	5/1/2023 3:19:10 PM	InProgress	Power BI Service	86392	345572	119.99	3.12%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 10:10:04 ...	5/1/2023 3:19:09 PM	InProgress	Power BI Service	86384	345540	119.98	3.12%
Uma BugBash_NB1 \ SynapseNotebook \ CapacityBugBash	Notebook Schedu...	4/26/2023 10:20:04 ...	5/1/2023 3:19:08 PM	InProgress	Power BI Service	86388	345557	119.99	3.12%
Total						3215497	11847875	4,113.91	107.13%

Next steps

- Install the Premium metrics app
- Use the Premium metrics app

Lakehouse and Delta Lake tables

Article • 05/23/2023

Microsoft Fabric Lakehouse is a data architecture platform for storing, managing, and analyzing structured and unstructured data in a single location. In order to achieve seamless data access across all compute engines in Microsoft Fabric, Delta Lake is chosen as the unified table format.

Saving data in the Lakehouse using capabilities such as Load to Tables or methods described in Options to get data into the Fabric Lakehouse, all data is saved in Delta format. Delta is also used as the default Spark table format mode in code-first experiences such as Notebooks and Spark Job Definitions.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

For a more comprehensive introduction to the Delta Lake table format, follow links in the Next steps section.

Big data, Apache Spark and legacy table formats

Microsoft Fabric Runtime for Apache Spark uses the same foundation as Azure Synapse Analytics Runtime for Apache Spark, but contain key differences to provide a more streamlined behavior across all engines in the Microsoft Fabric service. In Microsoft Fabric, key performance features are turned on by default. Advanced Apache Spark users may revert configurations to previous values to better align with specific scenarios.

Microsoft Fabric Lakehouse and the Apache Spark engine support all table types, both managed and unmanaged; this includes views and regular non-Delta Hive table formats. Tables defined using PARQUET, CSV, AVRO, JSON, and any Apache Hive compatible file format work as expected.

The Lakehouse explorer user interface experience varies depending on table type. Currently, the Lakehouse explorer only renders table objects.

Configuration differences with Azure Synapse Analytics

The following table contains the configuration differences between Azure Synapse Analytics and Microsoft Fabric Runtime for Apache Spark.

Apache Spark configuration	Microsoft Fabric value	Azure Synapse Analytics value	Notes
spark.sql.sources.default	delta	parquet	Default table format
spark.sql.parquet.vorder.enabled	true	N/A	V-Order writer
spark.sql.parquet.vorder.dictionaryPageSize	2 GB	N/A	Dictionary page size limit for V-Order
spark.microsoft.delta.optimizeWrite.enabled	true	unset (false)	Optimize Write

Auto discovery of tables

The Lakehouse explorer provides a tree-like view of the objects in the Microsoft Fabric Lakehouse item. It has a key capability of discovering and displaying tables that are described in the metadata repository and in OneLake storage. The table references are displayed under the `Tables` section of the Lakehouse explorer user interface. Auto discovery also applies to tables defined over OneLake shortcuts.

Tables over shortcuts

Microsoft Fabric Lakehouse supports tables defined over OneLake shortcuts, to provide utmost compatibility and no data movement. The following table contains the scenario best-practices for each item type when using it over shortcuts.

Shortcut destination	Where to create the shortcut	Best practice
Delta Lake table	`Tables` section	If multiple tables are present in the destination, create one shortcut per table.

Shortcut destination	Where to create the shortcut	Best practice
Folders with files	`Files` section	Use Apache Spark to use the destination directly using relative paths. Load the data into Lakehouse native Delta tables for maximum performance.
Legacy Apache Hive tables	`Files` section	Use Apache Spark to use the destination directly using relative paths, or create a metadata catalog reference using `CREATE EXTERNAL TABLE` syntax. Load the data into Lakehouse native Delta tables for maximum performance.

Load to Tables

Microsoft Fabric Lakehouse provides a convenient and productive user interface to streamline loading data into Delta tables. The Load to Tables feature allows a visual experiences to load common file formats and folders to Delta to boost analytical productivity to all personas. To learn more about the Load to Tables feature in details, read the Lakehouse Load to Tables reference documentation.

Delta Lake table optimization

Keeping tables in shape for the broad scope of analytics scenarios is no minor feat. Microsoft Fabric Lakehouse pro-actively enables the important parameters to minimize common problems associated with big data tables, such as compaction and small file sizes, and to maximize query performance. Still, there are many scenarios where those parameters need changes. The Delta Lake table optimization and V-Order article covers some key scenarios and provides an in-depth guide on how to efficiently maintain Delta tables for maximum performance.

Next steps

- What is Delta Lake?
- Delta Lake overview
- Shortcuts
- Load to Tables
- Spark workspace administration settings
- What is a Runtime in Fabric?

Load to Delta Lake tables

Article • 05/23/2023

Microsoft Fabric Lakehouse provides a feature to efficiently load common file types to an optimized Delta table ready for analytics. This guide describes the **Load to Tables** feature and its capabilities.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Summary of **Load to Tables** capabilities:

- Load single file into a new or existing table.
- Tables are loaded using the Delta Lake table format with V-Order optimization.

Supported file types

Load to Tables supports PARQUET and CSV file types. File extension case doesn't matter.

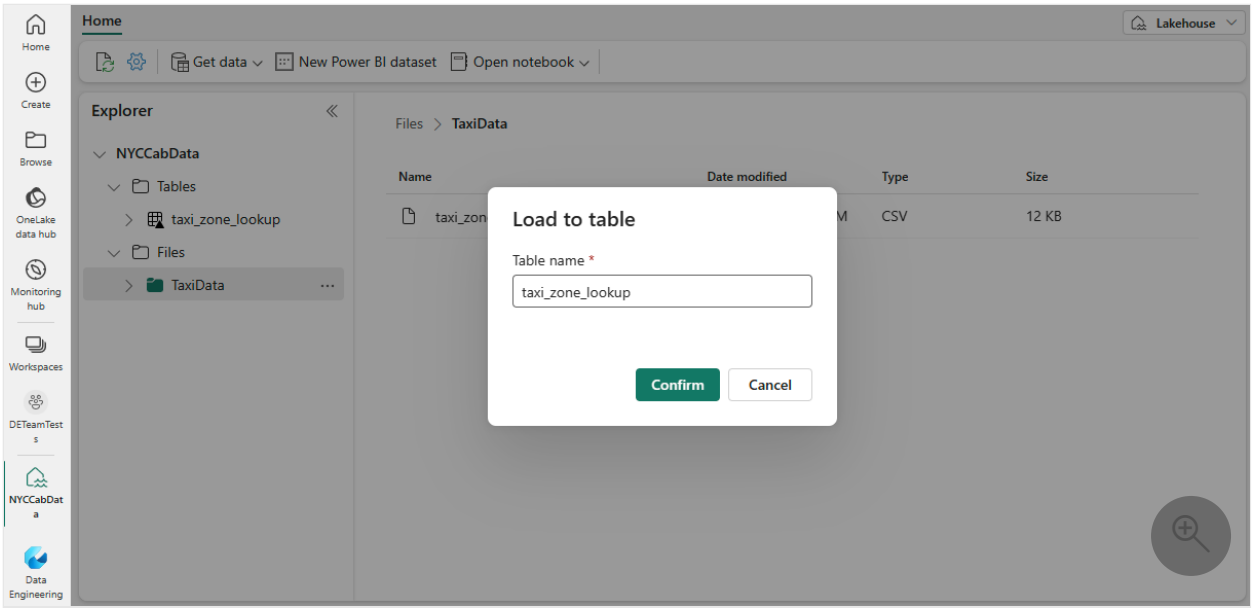
Table and column name validation and rules

The following standard applies to the Load to Delta experience:

- Table names can only contain alphanumeric characters and underscores. It may contain any English letter, upper or lower case, and underbar (_), with length up to 256 characters. No dashes (`-`) or space characters are allowed.
- Text files without column headers are replaced with standard `col#` notation as the table column names.
- Column names are validated during the load action. The Load to Delta algorithm replaces forbidden values with underbar (`_`). If no proper column name is achieved during validation, the load action fails. Column names allow any English letter, upper or lower case, underbar (_), and characters in other language such as Chinese in UTF, length up to 32 characters.

File load to Delta table

A file selected in the Lakehouse `Files` section to be loaded into a new Delta table in the `Tables` section. If the table already exists, it is dropped and then created.



Next steps

- What is Delta Lake?
- CSV file upload to Delta for Power BI reporting

Delta Lake table optimization and V-Order

Article • 05/23/2023

The Lakehouse and the Delta Lake table format are central to Microsoft Fabric, assuring that tables are optimized for analytics is a key requirement. This guide covers Delta Lake table optimization concepts, configurations and how to apply it to most common Big Data usage patterns.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

What is V-Order?

V-Order is a write time optimization to the parquet file format that enables lightning-fast reads under the Microsoft Fabric compute engines, such as Power BI, SQL, Spark and others.

Power BI and SQL engines make use of Microsoft Verti-Scan technology and V-Ordered parquet files to achieve in-memory like data access times. Spark and other non-Verti-Scan compute engines also benefit from the V-Ordered files with an average of 10% faster read times, with some scenarios up to 50%.

V-Order works by applying special sorting, row group distribution, dictionary encoding and compression on parquet files, thus requiring less network, disk, and CPU resources in compute engines to read it, providing cost efficiency and performance. V-Order sorting has a 15% impact on average write times but provides up to 50% more compression.

Its **100% open-source parquet format compliant**; all parquet engines can read it as a regular parquet files. Delta tables are more efficient than ever; features such as Z-Order are compatible with V-Order. Table properties and optimization commands can be used on control V-Order on its partitions.

V-Order is applied at the parquet file level. Delta tables and its features, such as Z-Order, compaction, vacuum, time travel, etc. are orthogonal to V-Order, as such, are

compatible and may be used together for extra benefits.

Controlling V-Order writes

V-Order is **enabled by default** in Microsoft Fabric and in Apache Spark it's controlled by the following configurations

Configuration	Default value	Description
spark.sql.parquet.vorder.enabled	true	Controls session level V-Order writing.
TBLPROPERTIES("delta.parquet.vorder.enabled")	false	Default V-Order mode on tables
Dataframe writer option: parquet.vorder.enabled	unset	Control V-Order writes using Dataframe writer

Use the following commands to control usage of V-Order writes.

Check V-Order configuration in Apache Spark session

Spark SQL

SQL

```
%%sql
GET spark.sql.parquet.vorder.enabled
```

Disable V-Order writing in Apache Spark session

Spark SQL

SQL

```
%%sql
SET spark.sql.parquet.vorder.enabled=FALSE
```

Enable V-Order writing in Apache Spark session

> **Important**
>
> When enabled at the session level. All parquet writes are made with V-Order enabled. This includes non-Delta parquet tables and Delta tables with the `parquet.vorder.enabled` table property set to either `true` or `false`.

Spark SQL

SQL

```
%%sql
SET spark.sql.parquet.vorder.enabled=TRUE
```

Control V-Order using Delta table properties

Enable V-Order table property during table creation:

SQL

```
%%sql
CREATE TABLE person (id INT, name STRING, age INT) USING parquet
TBLPROPERTIES("delta.parquet.vorder.enabled","true");
```

> **Important**
>
> When the table property is set to true; INSERT, UPDATE and MERGE commands will behave as expected and perform. If the V-Order session configuration is set to true or the spark.write enables it, then the writes will be V-Order even if the TBLPROPERTIES is set to false.

Enable or disable V-Order by altering the table property:

SQL

```
%%sql
ALTER TABLE person SET TBLPROPERTIES("delta.parquet.vorder.enabled","true");

ALTER TABLE person SET TBLPROPERTIES("delta.
parquet.vorder.enabled","false");

ALTER TABLE person UNSET TBLPROPERTIES("delta.parquet.vorder.enabled");
```

After you enable or disable V-Order using table properties, only future writes to the table are affected. Parquet files keep the ordering used when it was created. To change the current physical structure to apply or remove V-Order, read the "Control V-Order when optimizing a table" section bellow.

Controlling V-Order directly on write operations

All Apache Spark write commands inherit the session setting if not explicit. All following commands write using V-Order by implicitly inheriting the session configuration.

Python

```
df_source.write\
  .format("delta")\
  .mode("append")\
  .saveAsTable("myschema.mytable")

DeltaTable.createOrReplace(spark)\
  .addColumn("id","INT")\
  .addColumn("firstName","STRING")\
  .addColumn("middleName","STRING")\
  .addColumn("lastName","STRING",comment="surname")\
  .addColumn("birthDate","TIMESTAMP")\
  .location("Files/people")\
  .execute()

df_source.write\
  .format("delta")\
  .mode("overwrite")\
  .option("replaceWhere","start_date >= '2017-01-01' AND end_date <= '2017-
01-31'")\
  .saveAsTable("myschema.mytable")
```

> **ⓘ Important**
>
> V-Order applies only files affected by the predicate.

In a session where `spark.sql.parquet.vorder.enabled` is unset or set to false, the following commands would write using V-Order:

Python

```
df_source.write\
  .format("delta")\
  .mode("overwrite")\
  .option("replaceWhere","start_date >= '2017-01-01' AND end_date <= '2017-
01-31'")\
```

```
  .option("parquet.vorder.enabled ","true")\
  .saveAsTable("myschema.mytable")

DeltaTable.createOrReplace(spark)\
  .addColumn("id","INT")\
  .addColumn("firstName","STRING")\
  .addColumn("middleName","STRING")\
  .addColumn("lastName","STRING",comment="surname")\
  .addColumn("birthDate","TIMESTAMP")\
  .option("parquet.vorder.enabled","true")\
  .location("Files/people")\
  .execute()
```

What is Optimized Write?

Analytical workloads on Big Data processing engines such as Apache Spark perform most efficiently when using standardized larger file sizes. The relation between the file size, the number of files, the number of Spark workers and its configurations, play a critical role on performance. Ingestion workloads into data lake tables may have the inherited characteristic of constantly writing lots of small files; this scenario is commonly known as the "small file problem".

Optimize Write is a Delta Lake on Microsoft Fabric and Azure Synapse Analytics feature in the Apache Spark engine that reduces the number of files written and aims to increase individual file size of the written data. The target file size may be changed per a workload requirements using configurations.

The feature is **enabled by default** in Microsoft Fabric Runtime for Apache Spark. To learn more about Optimize Write usage scenarios, read the article The need for optimize write on Apache Spark

Merge optimization

Delta Lake MERGE command allows users to update a delta table with advanced conditions. It can update data from a source table, view or DataFrame into a target table by using MERGE command. However, the current algorithm in the open source distribution of Delta Lake isn't fully optimized for handling unmodified rows. The Microsoft Spark Delta team implemented a custom Low Shuffle Merge optimization, unmodified rows are excluded from an expensive shuffling operation that is needed for updating matched rows.

The implementation is controlled by the `spark.microsoft.delta.merge.lowShuffle.enabled` configuration, **enabled by default** in

the runtime. It requires no code changes and is fully compatible with the open-source distribution of Delta Lake. To learn more about Low Shuffle Merge usage scenarios, read the article Low Shuffle Merge optimization on Delta tables

Delta table maintenance

As Delta tables change, performance and storage cost efficiency tend to degrade for the following reasons:

- New data added to the table may skew data
- Batch and streaming data ingestion rates might bring in many small files
- Update and delete operations eventually create read overhead; parquet files are immutable by design, so Delta tables adds new parquet files which the changeset, further amplifying the issues imposed by the first two items.
- No longer needed data files and log files available in the storage.

In order to keep the tables at the best state for best performance, perform bin-compaction and vacuuming operations in the Delta tables. Bin-compaction is achieved by the OPTIMIZE command; it merges all changes into bigger, consolidated parquet files. Dereferenced storage clean-up is achieved by the VACUUM command.

> **Important**
>
> Properly designing the table physical structure based on the ingestion frequency and expected read patterns is likely more important than running the optimization commands described in this section.

Control V-Order when optimizing a table

The following command structures bin-compact and rewrite all affected files using V-Order, independent of the TBLPROPERTIES setting or session configuration setting:

SQL

```
%%sql
OPTIMIZE <table|fileOrFolderPath> VORDER;

OPTIMIZE <table|fileOrFolderPath> WHERE <predicate> VORDER;

OPTIMIZE <table|fileOrFolderPath> WHERE <predicate> [ZORDER  BY (col_name1,
col_name2, ...)] VORDER;
```

When ZORDER and VORDER are used together, Apache Spark performs bin-compaction, ZORDER, VORDER sequentially.

The following commands bin-compact and rewrite all affected files using the TBLPROPERTIES setting. If TBLPROPERTIES is set true to V-Order, all affected files are written as V-Order. If TBLPROPERTIES is unset or set to false to V-Order, it inherits the session setting; so in order to remove V-Order from the table, set the session configuration to false.

SQL

```
%%sql
OPTIMIZE <table|fileOrFolderPath>;

OPTIMIZE <table|fileOrFolderPath> WHERE predicate;

OPTIMIZE <table|fileOrFolderPath> WHERE predicate [ZORDER BY (col_name1,
col_name2, ...)];
```

Next steps

- What is Delta Lake?
- Lakehouse and Delta Lake
- The need for optimize write on Apache Spark
- Low Shuffle Merge optimization on Delta tables

How to use Microsoft Fabric notebooks

Article • 05/23/2023

Microsoft Fabric notebook is a primary code item for developing Apache Spark jobs and machine learning experiments, it's a web-based interactive surface used by data scientists and data engineers to write code benefiting from rich visualizations and Markdown text. Data engineers write code for data ingestion, data preparation, and data transformation. Data scientists also use notebooks to build machine learning solutions, including creating experiments and models, model tracking, and deployment.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

With a Microsoft Fabric notebook, you can:

- Get started with zero setup effort.
- Easily explore and process data with intuitive low-code experience.
- Keep data secure with built-in enterprise security features.
- Analyze data across raw formats (CSV, txt, JSON, etc.), processed file formats (parquet, Delta Lake, etc.), leveraging powerful Spark capabilities.
- Be productive with enhanced authoring capabilities and built-in data visualization.

This article describes how to use notebooks in data science and data engineering experiences.

Create notebooks

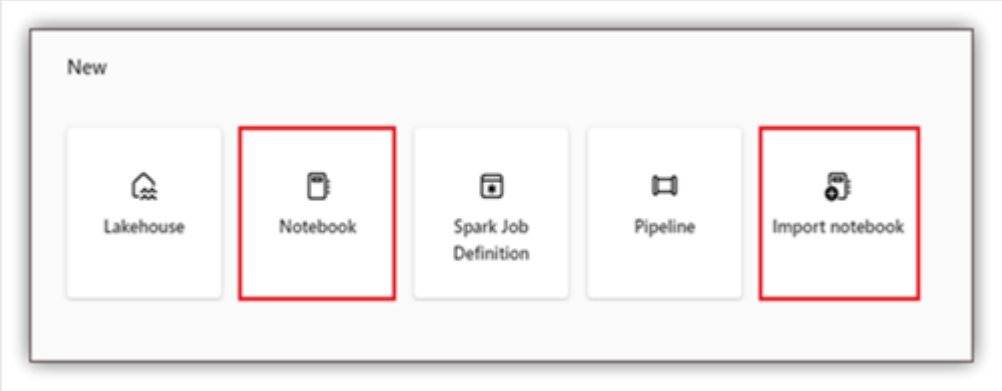
You can either create a new notebook or import an existing notebook.

Create a new notebook

Similar with other standard Microsoft Fabric item creation, you can easily create a new notebook from the Microsoft Fabric **Data Engineering** homepage, the workspace **New** button, or the **Create Hub**.

Import existing notebooks

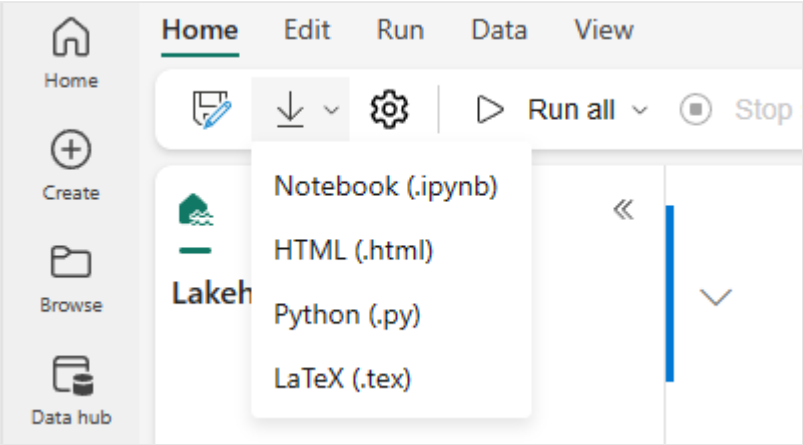
You can import one or more existing notebooks from your local computer to a Microsoft Fabric workspace from the **Data Engineering or the Data Science** homepage. Microsoft Fabric notebooks can recognize the standard Jupyter Notebook .ipynb files, and source files like .py, .scala, and .sql, and create new notebook items accordingly.



Export a notebook

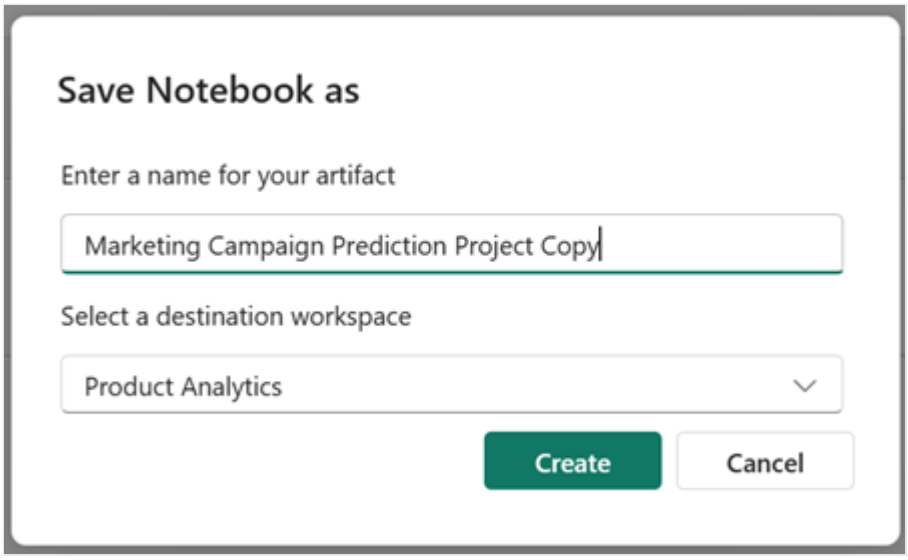
You can Export your notebook to other standard formats. Synapse notebook supports to be exported into:

- Standard Notebook file(.ipynb) that is usually used for Jupyter notebooks.
- HTML file(.html) that can be opened from browser directly.
- Python file(.py).
- Latex file(.tex).

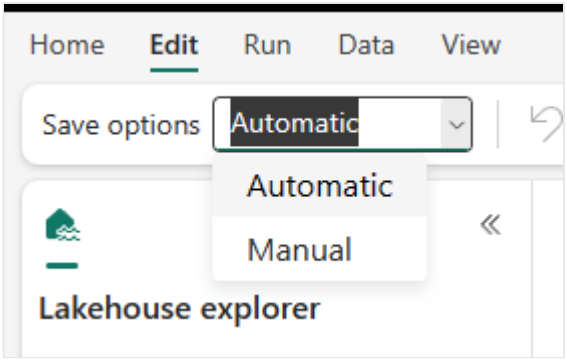


Save a notebook

In Microsoft Fabric, a notebook will by default save automatically after you open and edit it; you don't need to worry about losing code changes. You can also use **Save a copy** to clone another copy in the current workspace or to another workspace.



If you prefer to save a notebook manually, you can also switch to "Manual save" mode to have a "local branch" of your notebook item, and use **Save** or **CTRL+s** to save your changes.

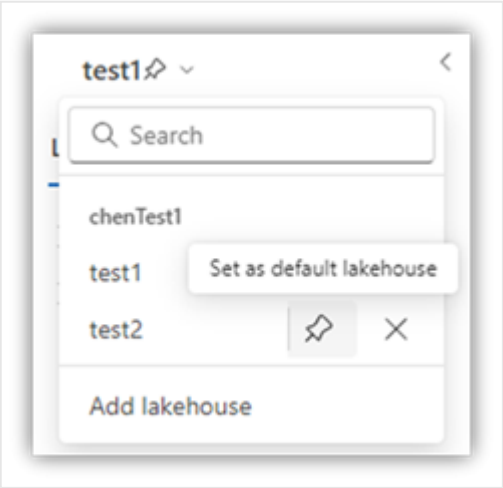


You can also switch to manual save mode from **Edit**->**Save options**->**Manual**. To turn on a local branch of your notebook then save it manually by clicking **Save** button or through "Ctrl" + "s" keybinding.

Connect lakehouses and notebooks

Microsoft Fabric notebook now supports interacting with lakehouses closely; you can easily add a new or existing lakehouse from the lakehouse explorer.

You can navigate to different lakehouses in the lakehouse explorer and set one lakehouse as the default by pinning it. It will then be mounted to the runtime working directory and you can read or write to the default lakehouse using a local path.



> **Note**
>
> You need to restart the session after pinning a new lakehouse or renaming the default lakehouse.

Add or remove a lakehouse

Selecting the **X** icon beside a lakehouse name removes it from the notebook tab, but the lakehouse item still exists in the workspace.

Select **Add lakehouse** to add more lakehouses to the notebook, either by adding an existing one or creating a new lakehouse.

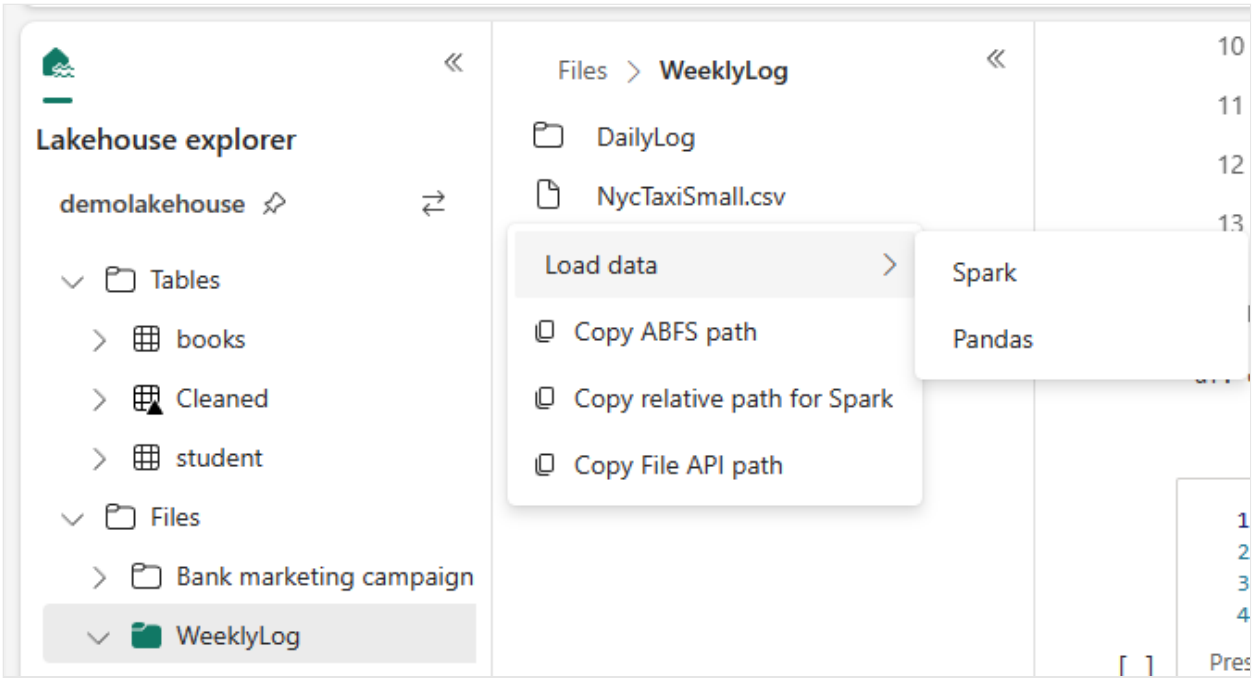
Explore a lakehouse file

The subfolder and files under the **Tables** and **Files** section of the **Lake** view appear in a content area between the lakehouse list and the notebook content. Select different folders in the **Tables** and **Files** section to refresh the content area.

Folder and File operations

If you select a file(.csv, .parquet, .txt, .jpg, .png, etc) with a right mouse click, both Spark and Pandas API are supported to load the data. A new code cell is generated and inserted to below of the focus cell.

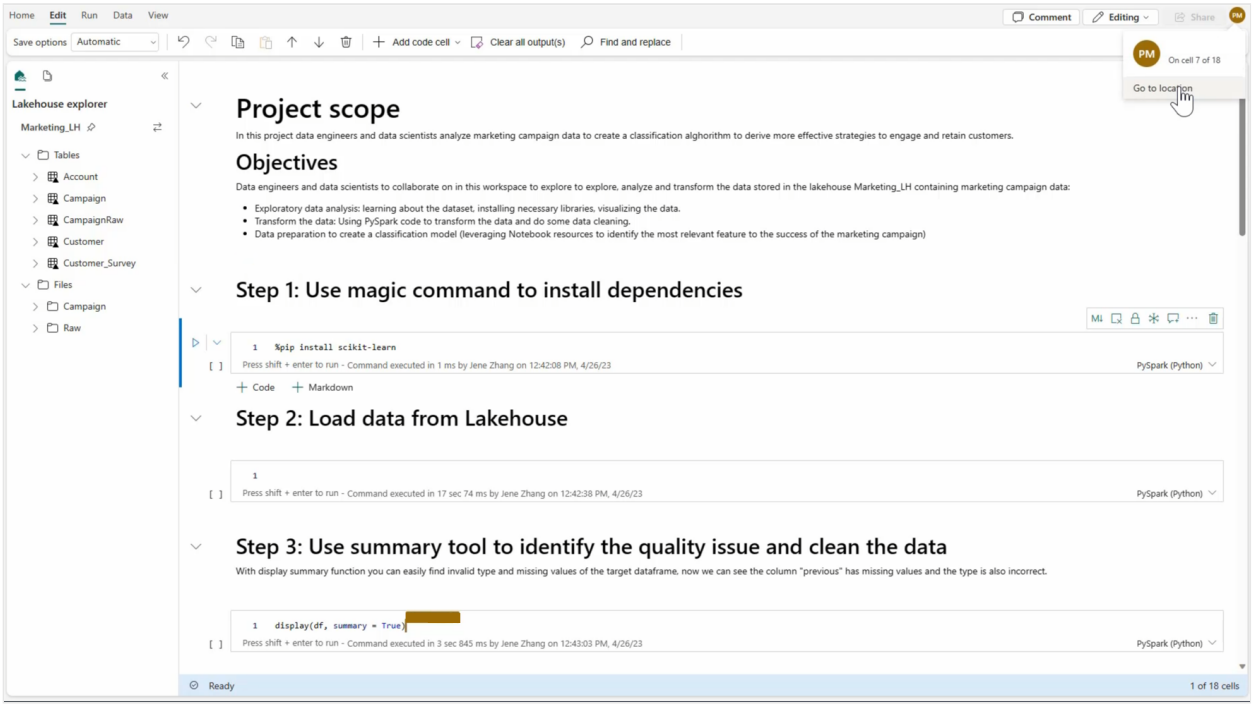
You can easily copy path with different format of the select file or folder and use the corresponding path in your code.



Collaborate in a notebook

The Microsoft Fabric notebook is a collaborative item that supports multiple users editing the same notebook.

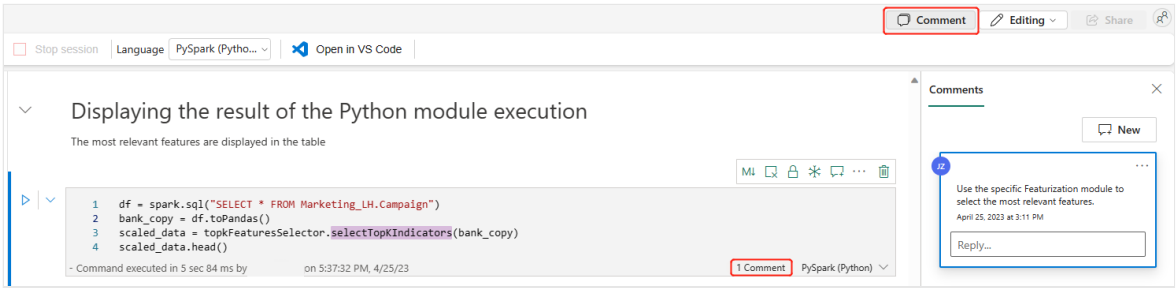
When you open a notebook, you enter the co-editing mode by default, and every notebook edit will be auto-saved. If your colleagues open the same notebook at the same time, you see their profile, run output, cursor indicator, selection indicator and editing trace. By leveraging the collaborating features, you can easily accomplish pair programming, remote debugging, and tutoring scenarios.



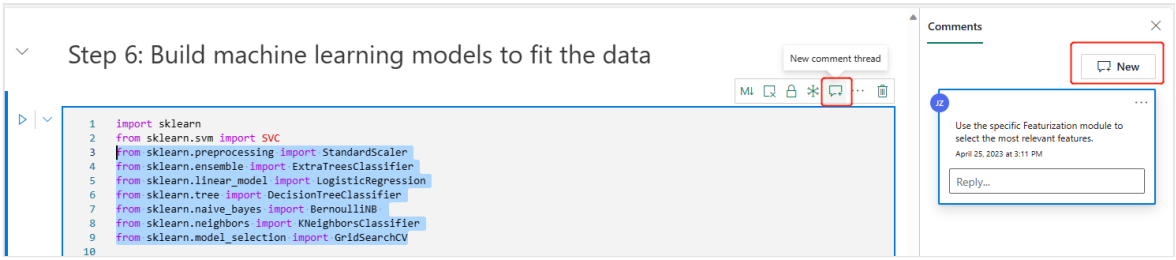
Comment a code cell

Commenting is another useful feature during collaborative scenarios. Currently, we support adding cell-level comments.

1. Select the **Comments** button on the notebook toolbar or cell comment indicator to open the **Comments** pane.



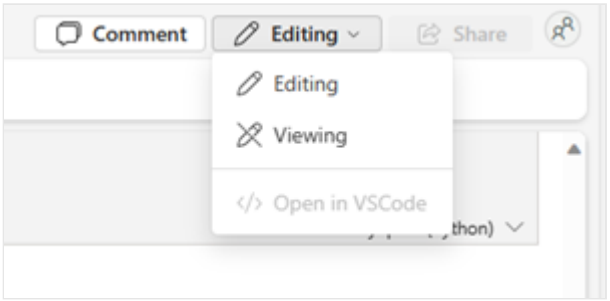
2. Select code in the code cell, select **New** in the **Comments** pane, add comments, and then select the post comment button to save.



3. You could perform **Edit comment**, **Resolve thread**, or **Delete thread** by selecting the More button besides your comment.

Switch Notebook mode

Fabric notebook support two modes for different scenarios, you can easily switch between **Editing** mode and **Viewing** mode.



- **Editing mode**: You can edit and run the cells and collaborate with others on the notebook.
- **Viewing mode**: You can only view the cell content, output, and comments of the notebook, all the operations that can lead to change the notebook will be

disabled.

Next steps

- Author and execute notebooks

Develop, execute, and manage Microsoft Fabric notebooks

Article • 05/23/2023

Microsoft Fabric notebook is a primary code item for developing Apache Spark jobs and machine learning experiments. It's a web-based interactive surface used by data scientists and data engineers to write code benefiting from rich visualizations and Markdown text. This article explains how to develop notebooks with code cell operations and run them.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Develop notebooks

Notebooks consist of cells, which are individual blocks of code or text that can be run independently or as a group.

We provide rich operations to develop notebooks:

- Add a cell
- Set a primary language
- Use multiple languages
- IDE-style IntelliSense
- Code snippets
- Drag and drop to insert snippets
- Drag and drop to insert images
- Format text cell with toolbar buttons
- Undo or redo cell operation
- Move a cell
- Delete a cell
- Collapse a cell input
- Collapse a cell output
- Lock or freeze a cell
- Notebook contents
- Markdown folding
- Find and replace

Add a cell

There are multiple ways to add a new cell to your notebook.

1. Hover over the space between two cells and select **Code** or **Markdown**.
2. Use Shortcut keys under command mode. Press **A** to insert a cell above the current cell. Press **B** to insert a cell below the current cell.

Set a primary language

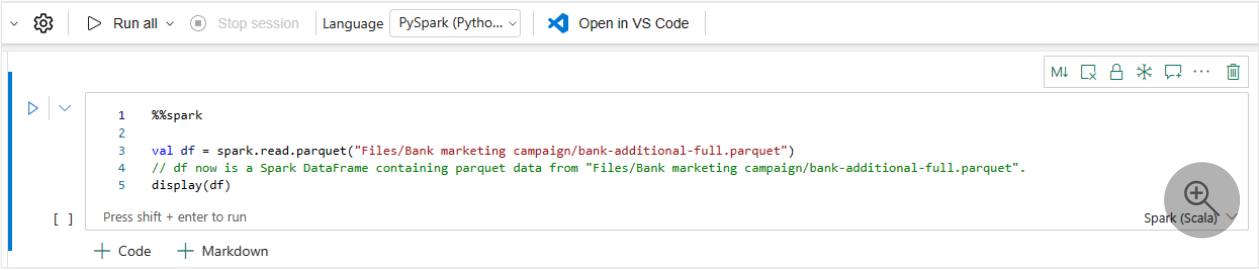
Microsoft Fabric notebooks currently support four Apache Spark languages:

- PySpark (Python)
- Spark (Scala)
- Spark SQL
- SparkR

You can set the primary language for new added cells from the dropdown list in the top command bar.

Use multiple languages

You can use multiple languages in a notebook by specifying the language magic command at the beginning of a cell, you can also switch the cell language from the language picker. The following table lists the magic commands to switch cell languages.



Magic command	Language	Description
%%pyspark	Python	Execute a **Python** query against Spark Context.
%%spark	Scala	Execute a **Scala** query against Spark Context.
%%sql	SparkSQL	Execute a **SparkSQL** query against Spark Context.
%%html	Html	Execute a **HTML** query against Spark Context.
%%sparkr	R	Execute a **R** query against Spark Context.

The following image is an example of how you can write a PySpark query using the **%%pyspark** magic command in a **Spark(Scala)** notebook. Notice that the primary language

for the notebook is set to PySpark.

IDE-style IntelliSense

Microsoft Fabric notebooks are integrated with the Monaco editor to bring IDE-style IntelliSense to the cell editor. Syntax highlight, error marker, and automatic code completions help you to write code and identify issues quicker.

The IntelliSense features are at different levels of maturity for different languages. The following table shows what's supported:

Languages	Syntax Highlight	Syntax Error Marker	Syntax Code Completion	Variable Code Completion	System Function Code Completion	User Function Code Completion	Smart Indent	Code Folding
PySpark (Python)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Spark (Scala)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SparkSQL	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
SparkR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

> **Note**
>
> An active Spark session is required to make use of the IntelliSense code completion.

Code snippets

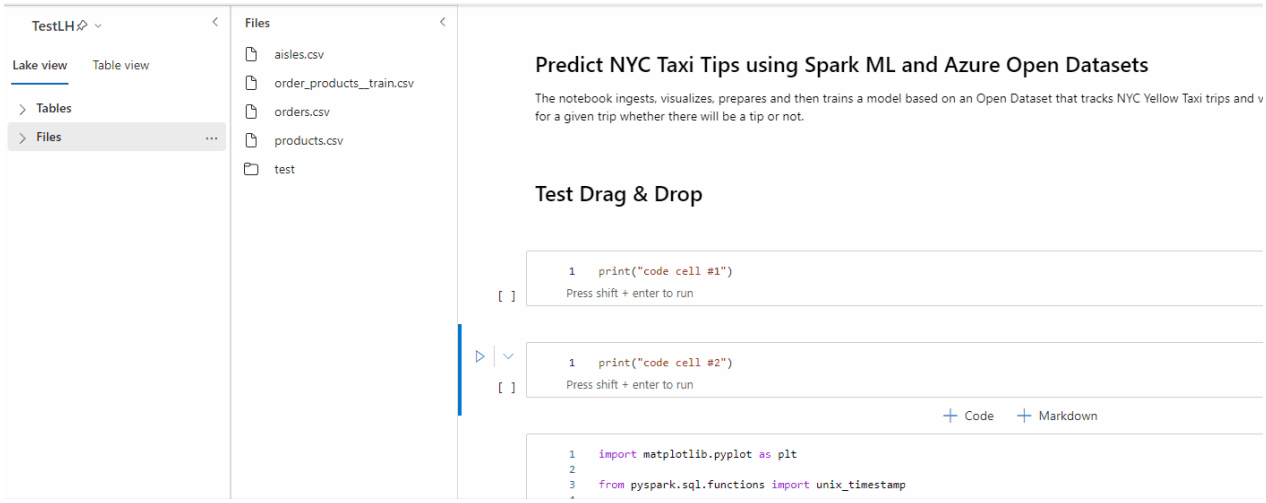
Microsoft Fabric notebooks provide code snippets that help you write commonly used code patterns easily, like:

- Reading data as a Spark DataFrame, or
- Drawing charts with Matplotlib.

Snippets appear in Shortcut keys of IDE style IntelliSense mixed with other suggestions. The code snippets contents align with the code cell language. You can see available snippets by typing **Snippet** or any keywords appear in the snippet title in the code cell editor. For example, by typing **read** you can see the list of snippets to read data from various data sources.

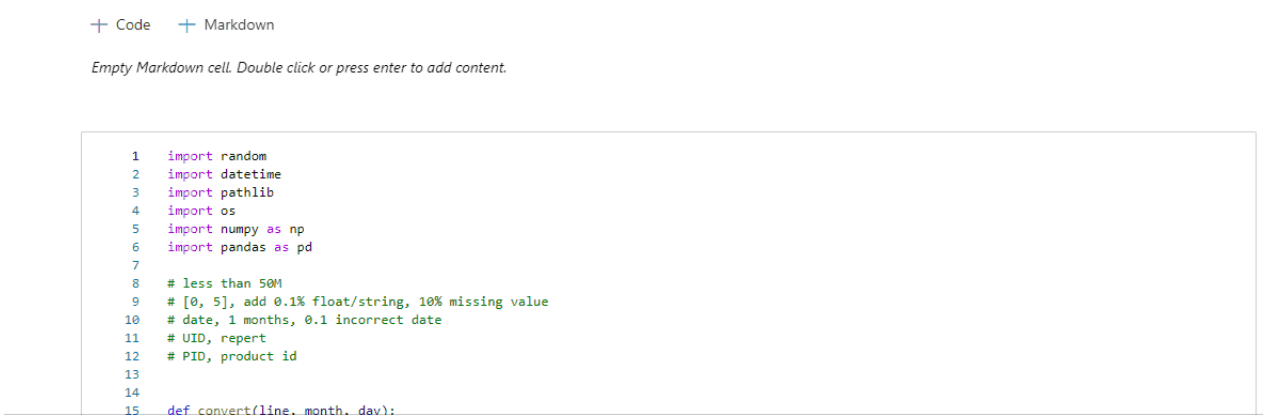
Drag and drop to insert snippets

You can use drag and drop to read data from Lakehouse explorer conveniently. Multiple file types are supported here, you can operate on text files, tables, images, etc. You can either drop to an existing cell or to a new cell. The notebook generates the code snippet accordingly to preview the data.



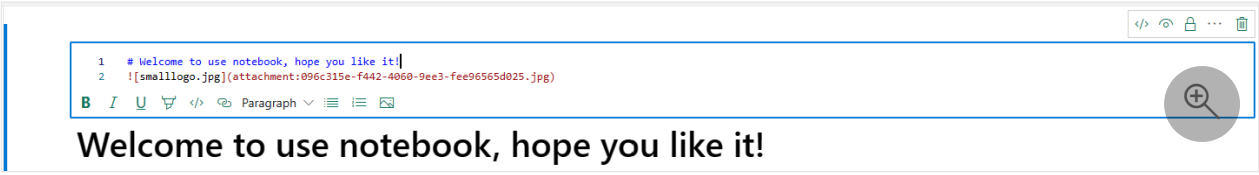
Drag and drop to insert images

You can use drag and drop to insert images from your browser or local computer to a markdown cell conveniently.



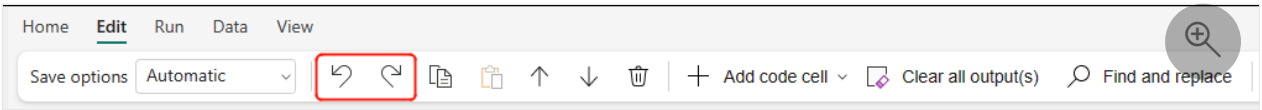
Format text cell with toolbar buttons

You can use the format buttons in the text cells toolbar to do common markdown actions.



Undo or redo cell operation

Select the **Undo** or **Redo** button, or press **Z** or **Shift+Z** to revoke the most recent cell operations. You can undo or redo up to the latest 10 historical cell operations.



Supported undo cell operations:

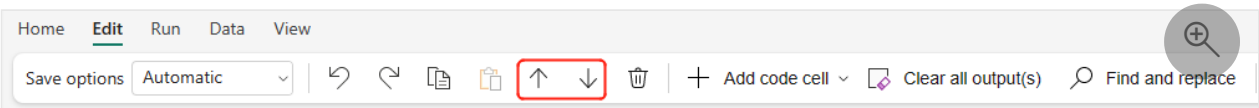
- Insert or delete cell: You could revoke the delete operations by selecting **Undo**, the text content is kept along with the cell.
- Reorder cell.
- Toggle parameter.
- Convert between code cell and Markdown cell.

> **Note**
>
> In-cell text operations and code cell commenting operations can't be undone. You can undo or redo up to the latest 10 historical cell operations.

Move a cell

You can drag from the empty part of a cell and drop it to the desired position.

You can also move the selected cell using **Move up** and **Move down** on the ribbon.



Delete a cell

To delete a cell, select the delete button at the right hand of the cell.

You can also use shortcut keys under command mode. Press **Shift+D** to delete the current cell.

Collapse a cell input

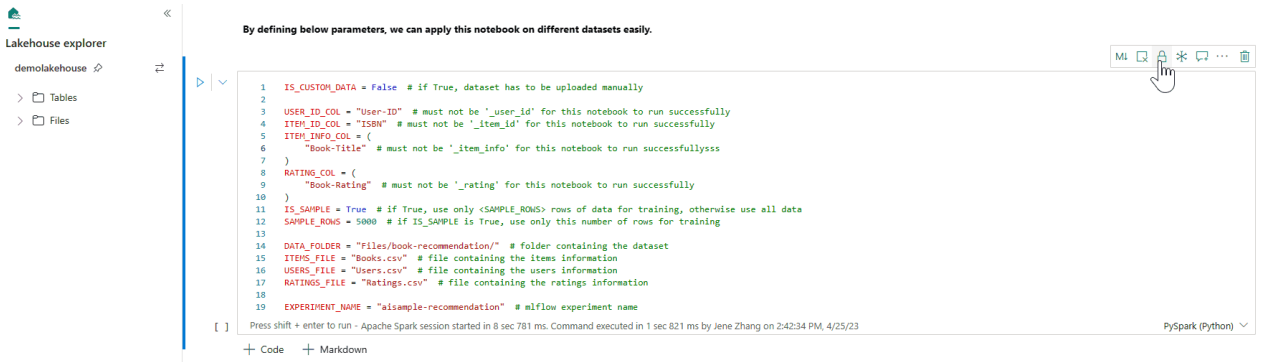
Select the **More commands** ellipses (...) on the cell toolbar and **Hide input** to collapse current cell's input. To expand it, Select the **Show input** while the cell is collapsed.

Collapse a cell output

Select the **More commands** ellipses (...) on the cell toolbar and **Hide output** to collapse current cell's output. To expand it, select the **Show output** while the cell's output is hidden.

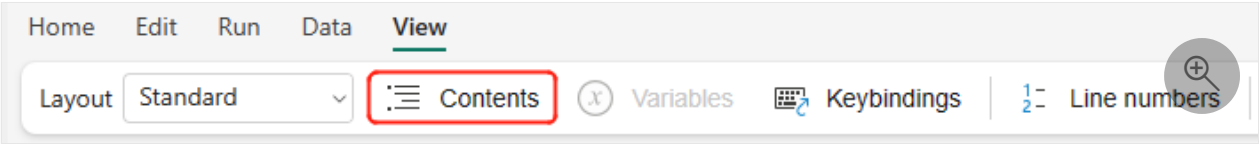
Lock or freeze a cell

Lock and freeze cell operations allow you to make cells read-only or stop code cells from being run on an individual cell basis.



Notebook contents

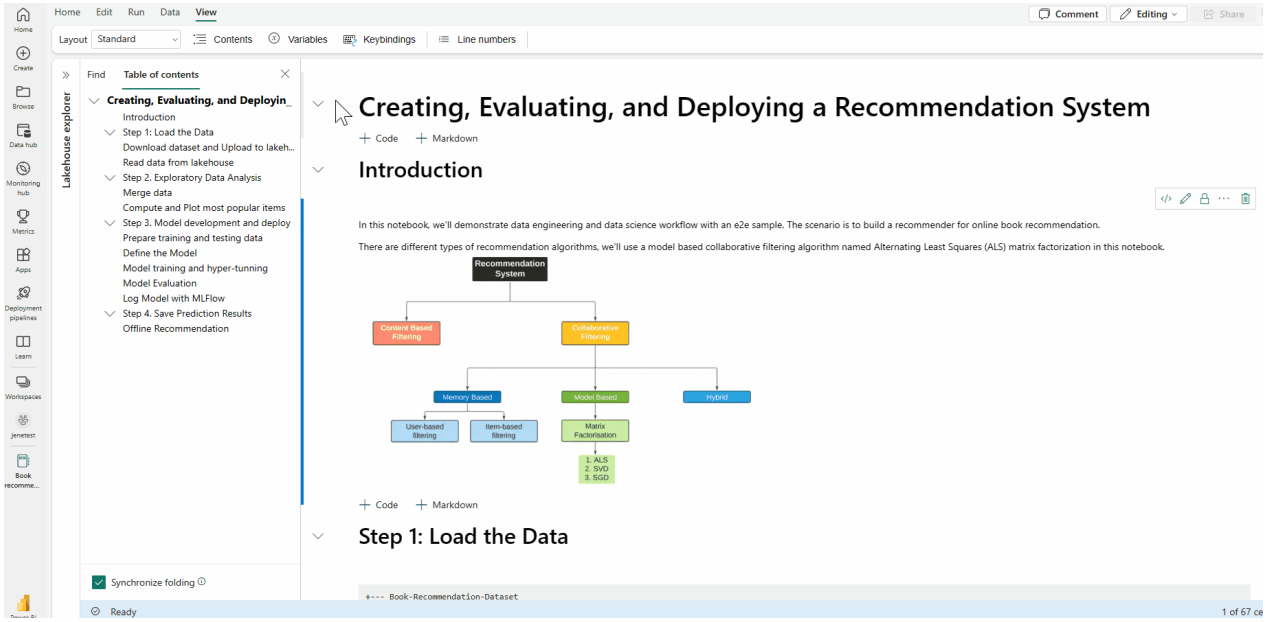
The Outlines or Table of Contents presents the first markdown header of any markdown cell in a sidebar window for quick navigation. The Outlines sidebar is resizable and collapsible to fit the screen in the best ways possible. You can select the **Contents** button on the notebook command bar to open or hide the sidebar.



Markdown folding

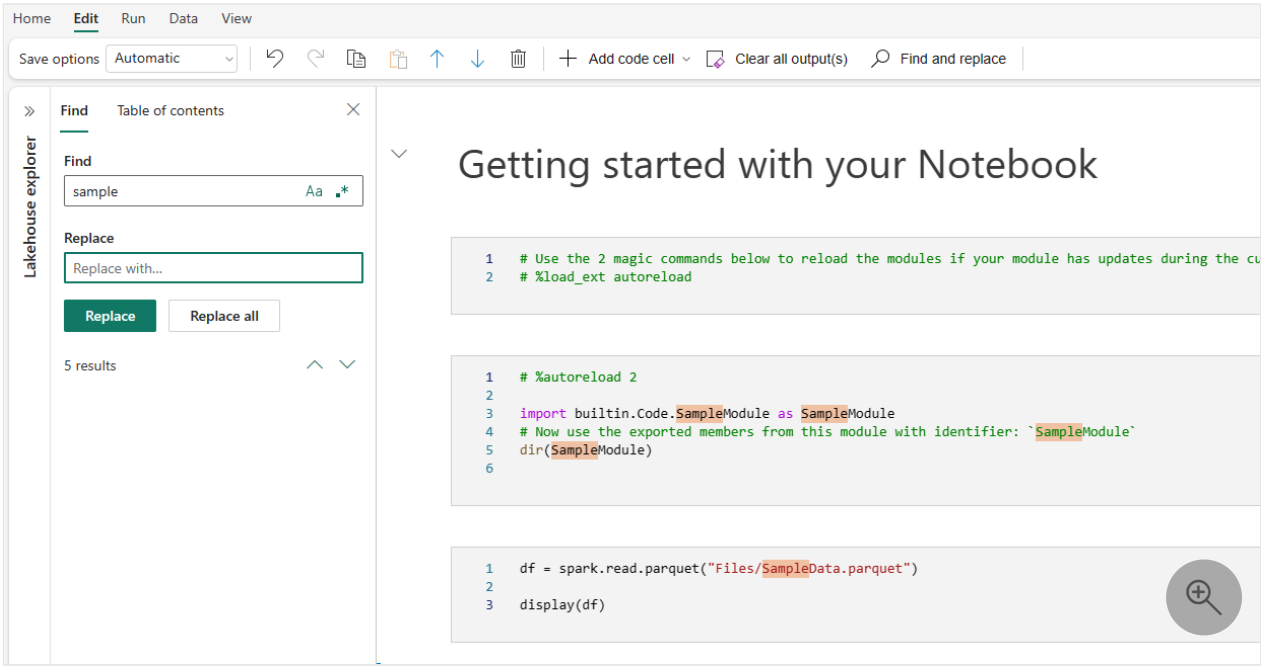
The markdown folding allows you to hide cells under a markdown cell that contains a heading. The markdown cell and its hidden cells are treated the same as a set of contiguous

multi-selected cells when performing cell operations.



Find and replace

Find and replace can help you easily match and locate the keywords or expression within your notebook content, and you can replace the target string with a new string.



Run notebooks

You can run the code cells in your notebook individually or all at once. The status and progress of each cell is represented in the notebook.

Run a cell

There are several ways to run the code in a cell.

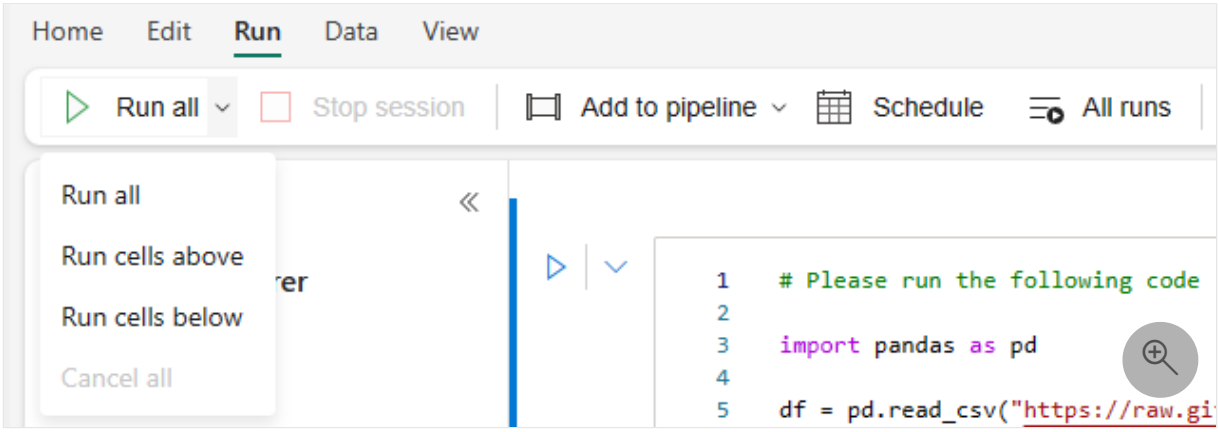
1. Hover on the cell you want to run and select the **Run Cell** button or press **Ctrl+Enter**.
2. Use Shortcut keys under command mode. Press **Shift+Enter** to run the current cell and select the next cell. Press **Alt+Enter** to run the current cell and insert a new cell.

Run all cells

Select the **Run All** button to run all the cells in the current notebook in sequence.

Run all cells above or below

Expand the dropdown list from **Run all** button, then select **Run cells above** to run all the cells above the current in sequence. Select **Run cells below** to run the current cell and all the cells below the current in sequence.

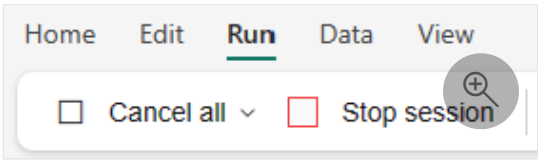


Cancel all running cells

Select the **Cancel All** button to cancel the running cells or cells waiting in the queue.

Stop session

Stop session cancels the running and waiting cells and stops the current session, you can restart a brand new session if you click the run button again.



Notebook reference run

Besides using mssparkutils reference run API You can also use `%run <notebook name>` magic command to reference another notebook within current notebook's context. All the variables defined in the reference notebook are available in the current notebook. `%run` magic

command supports nested calls but not support recursive calls. You'll receive an exception if the statement depth is larger than **five**.

Example: `%run Notebook1 { "parameterInt": 1, "parameterFloat": 2.5, "parameterBool": true, "parameterString": "abc" }`.

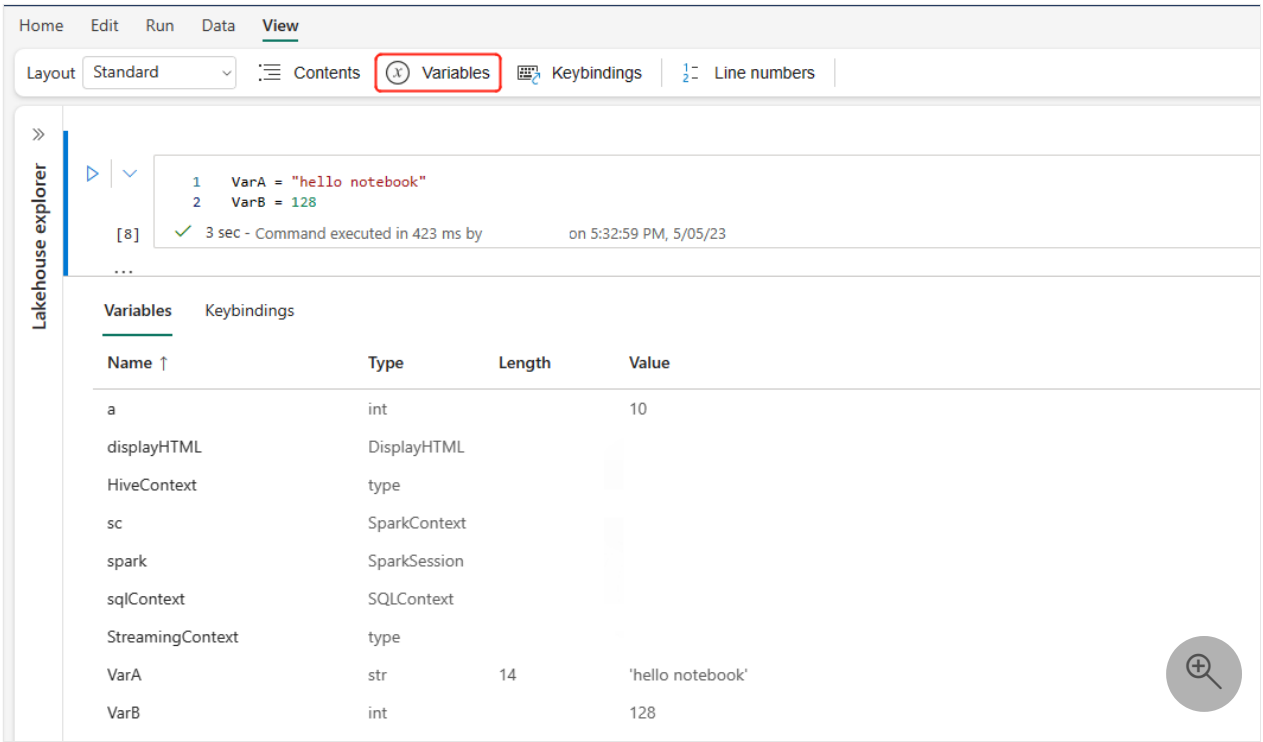
Notebook reference works in both interactive mode and pipeline.

> **Note**
>
> - `%run` command currently only supports reference notebooks that in the same workspace with current notebook.
> - `%run` command currently only supports to 4 parameter value types: `int`, `float`, `bool`, `string`, variable replacement operation is not supported.
> - `%run` command do not support nested reference that depth is larger than **five**.

Variable explorer

Microsoft Fabric notebook provides a built-in variables explorer for you to see the list of the variables name, type, length, and value in the current Spark session for PySpark (Python) cells. More variables show up automatically as they're defined in the code cells. Clicking on each column header sorts the variables in the table.

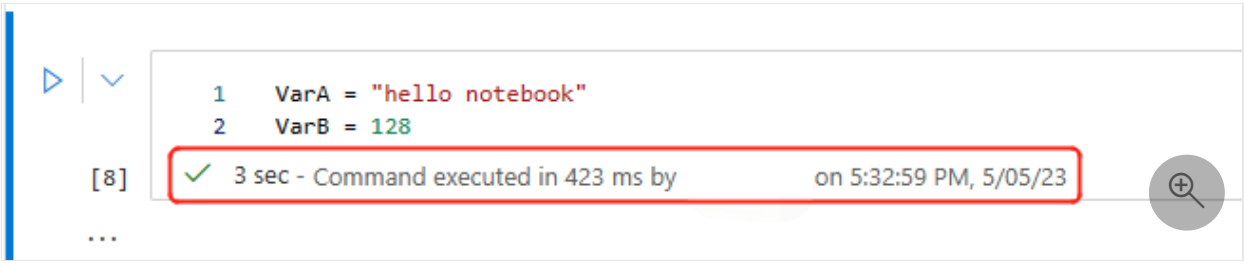
You can select the **Variables** button on the notebook ribbon "View" tab to open or hide the variable explorer.



> **Note**
>
> Variable explorer only supports python.

Cell status indicator

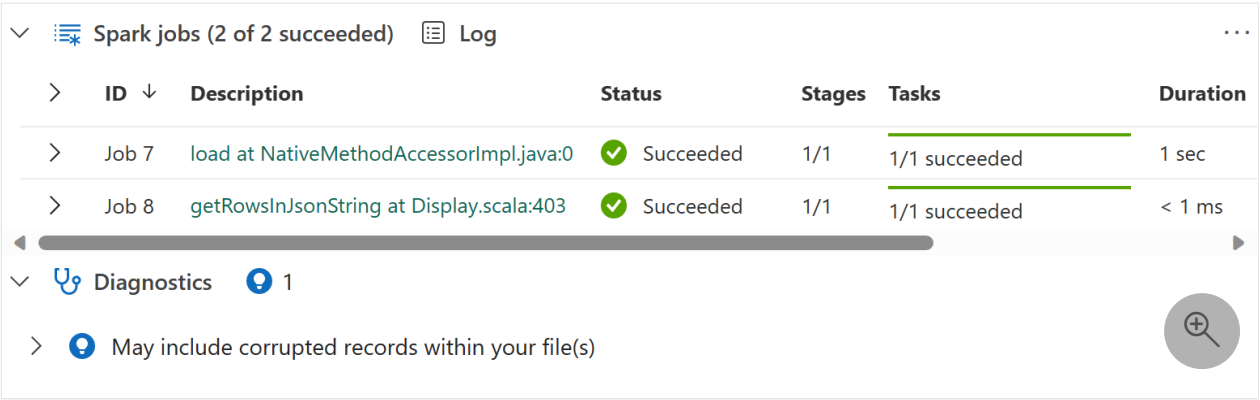
A step-by-step cell execution status is displayed beneath the cell to help you see its current progress. Once the cell run is complete, an execution summary with the total duration and end time is shown and stored there for future reference.



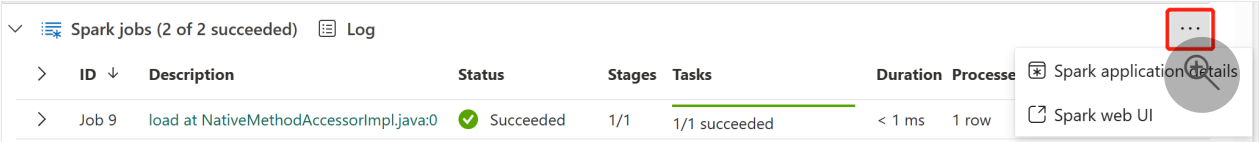
Inline spark job indicator

The Microsoft Fabric notebook is Spark based. Code cells are executed on the spark cluster remotely. A Spark job progress indicator is provided with a real-time progress bar appears to help you understand the job execution status. The number of tasks per each job or stage help you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) via selecting the link on the job (or stage) name.

You can also find the **Cell level real-time log** next to the progress indicator, and **Diagnostics** can provide you with useful suggestions to help refine and debug the code.

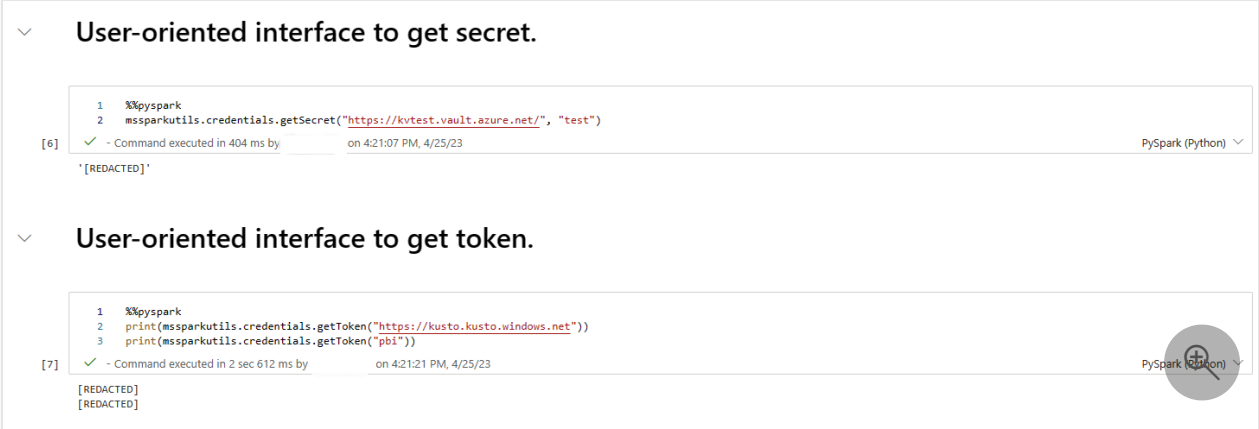


In **More actions**, you can easily navigate to the **Spark application details** page and **Spark web UI** page.



Secret redaction

To prevent the credentials being accidentally leaked when running notebooks, Fabric notebook support **Secret redaction** to replace the secret values that are displayed in the cell output with `[REDACTED]`, Secret redaction is applicable for **Python**, **Scala** and **R**.



Magic commands in notebook

Built-in magics

You can use familiar Ipython magic commands in Fabric notebooks. Review the following list as the current available magic commands.

> **Note**
>
> Only following magic commands are supported in Fabric pipeline : %%pyspark, %%spark, %%csharp, %%sql.

Available line magics: %lsmagic, %time, %timeit, %history, %run, %load, %alias, %alias_magic, %autoawait, %autocall, %automagic, %bookmark, %cd, %colors, %dhist, %dirs, %doctest_mode, %killbgscripts, %load_ext, %logoff, %logon, %logstart, %logstate, %logstop, %magic, %matplotlib, %page, %pastebin, %pdef, %pfile, %pinfo, %pinfo2, %popd, %pprint, %precision, %prun, %psearch, %psource, %pushd, %pwd, %pycat, %quickref, %rehashx, %reload_ext, %reset, %reset_selective, %sx, %system, %tb, %unalias, %unload_ext, %who, %who_ls, %whos, %xdel, %xmode.

Fabric notebook also supports improved library management commands **%pip**, **%conda**, check Manage Apache Spark libraries in Microsoft Fabric for the usage.

Available cell magics: %%time, %%timeit, %%capture, %%writefile, %%sql, %%pyspark, %%spark, %%csharp, %%html, %%bash, %%markdown, %%perl, %%script, %%sh.

Custom magics

You can also build out more custom magic commands to meet your specific needs as the below example shows.

1. Create a notebook with name "*MyLakehouseModule*".

Register My magic command with IPython in Notebook: MyLakehouseModule

```
@magics_class
class MyLakeHouseMagic(Magics):
    @line_magic
    def list_lh(self):
        "list all my lakehouse account"
        # do operations
        return ['LH1', 'LH2']

    @line_magic
    def create_lh(self, name):
        "create lakehouse with name"
        # do operations
        return "create Lakehouse with name: " + name
```

Press shift + enter to run

2. In another notebook reference the "*MyLakehouseModule*" and its magic commands, by this way you can organize your project with notebooks that using different languages conveniently.

```
%run MyLakehouseModule
```

Press shift + enter to run

Use it everywhere, in every language

```
%create_lh "mylakehouse"
```

✓ 20 sec - Command executed in 9 sec 973 ms by mgr nbs on 4:32:09 PM, 6/23/22

```
'create Lakehouse with name: "mylakehouse"'
```

IPython Widgets

IPython Widgets are eventful python objects that have a representation in the browser. You can use IPython Widgets as low-code controls (for example, slider, text box) in your notebook just like the Jupyter notebook, currently it only works in Python context.

To use IPython Widget

1. You need to import *ipywidgets* module first to use the Jupyter Widget framework.

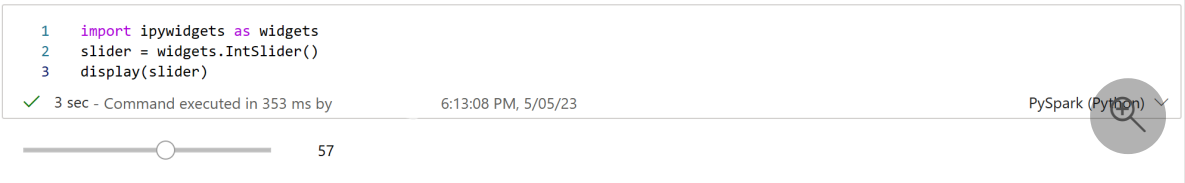
```
Python

import ipywidgets as widgets
```

2. You can use top-level *display* function to render a widget, or leave an expression of *widget* type at the last line of code cell.

```
Python

slider = widgets.IntSlider()
display(slider)
```

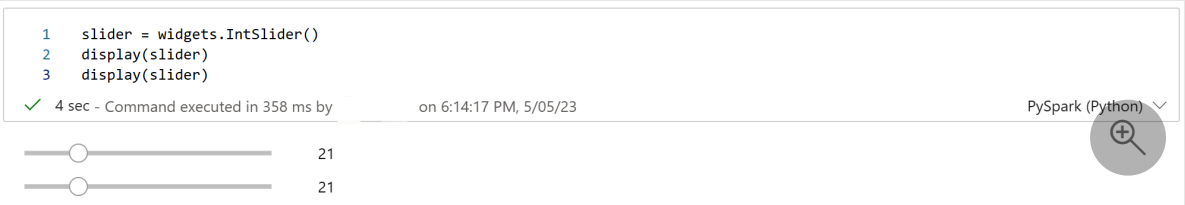
3. Run the cell, the widget displays in the output area.

```
Python

slider = widgets.IntSlider()
display(slider)
```



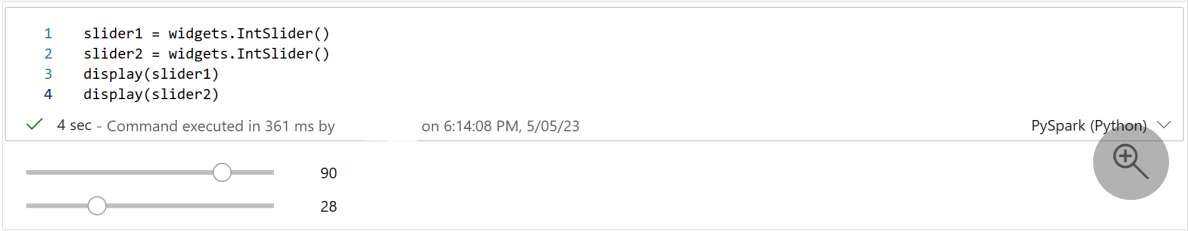
4. You can use multiple *display()* calls to render the same widget instance multiple times, but they remain in sync with each other.

```
Python

slider = widgets.IntSlider()
display(slider)
display(slider)
```



5. To render two widgets independent of each other, create two widget instances:

```
Python

slider1 = widgets.IntSlider()
slider2 = widgets.IntSlider()
display(slider1)
display(slider2)
```



Supported widgets

Widgets type	Widgets
Numeric widgets	IntSlider, FloatSlider, FloatLogSlider, IntRangeSlider, FloatRangeSlider, IntProgress, FloatProgress, BoundedIntText, BoundedFloatText, IntText, FloatText
Boolean widgets	ToggleButton, Checkbox, Valid
Selection widgets	Dropdown, RadioButtons, Select, SelectionSlider, SelectionRangeSlider, ToggleButtons, SelectMultiple
String Widgets	Text, Text area, Combobox, Password, Label, HTML, HTML Math, Image, Button
Play (Animation) widgets	Date picker, Color picker, Controller
Container or Layout widgets	Box, HBox, VBox, GridBox, Accordion, Tabs, Stacked

Known limitations

1. The following widgets aren't supported yet, you could follow the corresponding workaround as follows:

Functionality	Workaround
Output widget	You can use *print()* function instead to write text into stdout.
widgets.jslink()	You can use *widgets.link()* function to link two similar widgets.
FileUpload widget	Not supported yet.

2. Global *display* function provided by Microsoft Fabric doesn't support displaying multiple widgets in one call (i.e. *display(a, b)*), which is different from IPython *display* function.

3. If you close a notebook that contains IPython Widget, you can't see or interact with it until you execute the corresponding cell again.

Python logging in Notebook

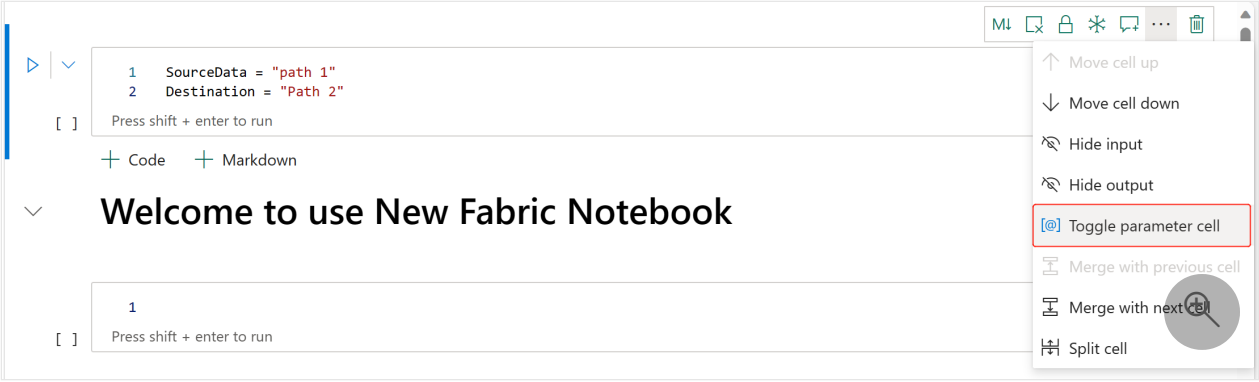
You can find Python logs and set different log levels and format like the sample code shown here:

```python
import logging

# Customize the logging format for all loggers
FORMAT = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
formatter = logging.Formatter(fmt=FORMAT)
for handler in logging.getLogger().handlers:
    handler.setFormatter(formatter)

# Customize log level for all loggers
logging.getLogger().setLevel(logging.INFO)

# Customize the log level for a specific logger
customizedLogger = logging.getLogger('customized')
customizedLogger.setLevel(logging.WARNING)

# logger that use the default global log level
defaultLogger = logging.getLogger('default')
defaultLogger.debug("default debug message")
defaultLogger.info("default info message")
defaultLogger.warning("default warning message")
defaultLogger.error("default error message")
defaultLogger.critical("default critical message")

# logger that use the customized log level
customizedLogger.debug("customized debug message")
customizedLogger.info("customized info message")
customizedLogger.warning("customized warning message")
customizedLogger.error("customized error message")
customizedLogger.critical("customized critical message")
```

Integrate a notebook

Designate a parameters cell

To parameterize your notebook, select the ellipses (...) to access the **more commands** at the cell toolbar. Then select **Toggle parameter cell** to designate the cell as the parameters cell.



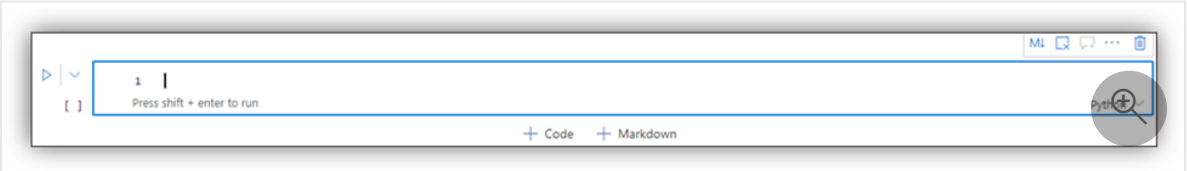
The parameter cell is useful for integrating notebook in pipeline, pipeline activity looks for the parameters cell and treats this cell as defaults for the parameters passed in at execution time. The execution engine adds a new cell beneath the parameters cell with input parameters in order to overwrite the default values.

Shortcut keys

Similar to Jupyter Notebooks, Microsoft Fabric notebooks have a modal user interface. The keyboard does different things depending on which mode the notebook cell is in. Microsoft Fabric notebooks support the following two modes for a given code cell: command mode and edit mode.

1. A cell is in command mode when there's no text cursor prompting you to type. When a cell is in Command mode, you can edit the notebook as a whole but not type into individual cells. Enter command mode by pressing ESC or using the mouse to select outside of a cell's editor area.

2. Edit mode can be indicated from a text cursor that prompting you to type in the editor area. When a cell is in edit mode, you can type into the cell. Enter edit mode by pressing Enter or using the mouse to select on a cell's editor area.



Shortcut keys under command mode

Action	Notebook shortcuts
Run the current cell and select below	Shift+Enter

Action	Notebook shortcuts
Run the current cell and insert below	Alt+Enter
Run current cell	Ctrl+Enter
Select cell above	Up
Select cell below	Down
Select previous cell	K
Select next cell	J
Insert cell above	A
Insert cell below	B
Delete selected cells	Shift + D
Switch to edit mode	Enter

Shortcut keys under edit mode

Using the following keystroke shortcuts, you can more easily navigate and run code in Microsoft Fabric notebooks when in Edit mode.

Action	Notebook shortcuts
Move cursor up	Up
Move cursor down	Down
Undo	Ctrl + Z
Redo	Ctrl + Y
Comment or Uncomment	Ctrl + /
Delete word before	Ctrl + Backspace
Delete word after	Ctrl + Delete
Go to cell start	Ctrl + Home
Go to cell end	Ctrl + End
Go one word left	Ctrl + Left
Go one word right	Ctrl + Right
Select all	Ctrl + A
Indent	Ctrl +]

Action	Notebook shortcuts
Dedent	Ctrl + [
Switch to command mode	Esc

You can easily find all shortcut keys from notebook ribbon *View* -> *Keybindings*.

Next steps

- Notebook visualization
- Introduction of Fabric MSSparkUtils

Notebook visualization in Microsoft Fabric

Article • 05/23/2023

Microsoft Fabric is an integrated analytics service that accelerates time to insight, across data warehouses and big data analytics systems. Data visualization in notebook is a key component in being able to gain insight into your data. It helps make big and small data easier for humans to understand. It also makes it easier to detect patterns, trends, and outliers in groups of data.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

When you use Apache Spark in Microsoft Fabric, there are various built-in options to help you visualize your data, including Microsoft Fabric notebook chart options, and access to popular open-source libraries.

Notebook chart options

When using a Microsoft Fabric notebook, you can turn your tabular results view into a customized chart using chart options. Here, you can visualize your data without having to write any code.

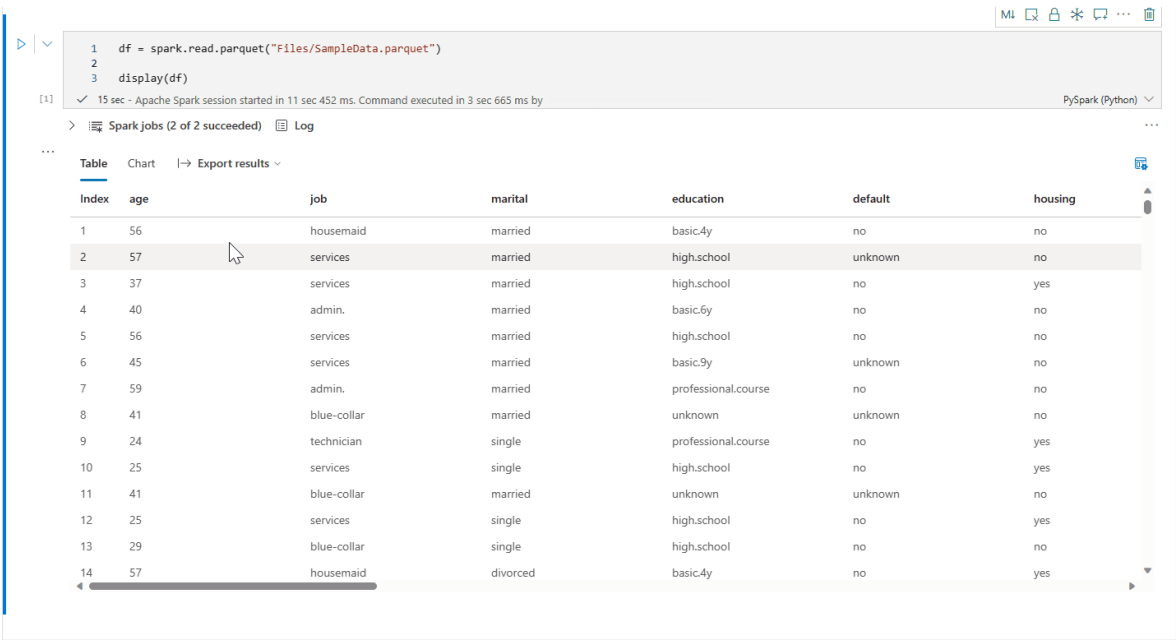
display(df) function

The *display* function allows you to turn SQL query result and Apache Spark dataframes into rich data visualizations. The *display* function can be used on dataframes created in PySpark and Scala.

To access the chart options:

1. The output of *%%sql* magic commands appear in the rendered table view by default. You can also call *display(df)* on Spark DataFrames or Resilient Distributed Datasets (RDD) function to produce the rendered table view.

2. Once you have a rendered table view, switch to the **Chart** view.



3. You can now customize your visualization by specifying the following values:

Configuration	Description
Chart Type	The display function supports a wide range of chart types, including bar charts, scatter plots, line graphs, and more.
Key	Specify the range of values for the x-axis.
Value	Specify the range of values for the y-axis values.
Series Group	Used to determine the groups for the aggregation.
Aggregation	Method to aggregate data in your visualization.

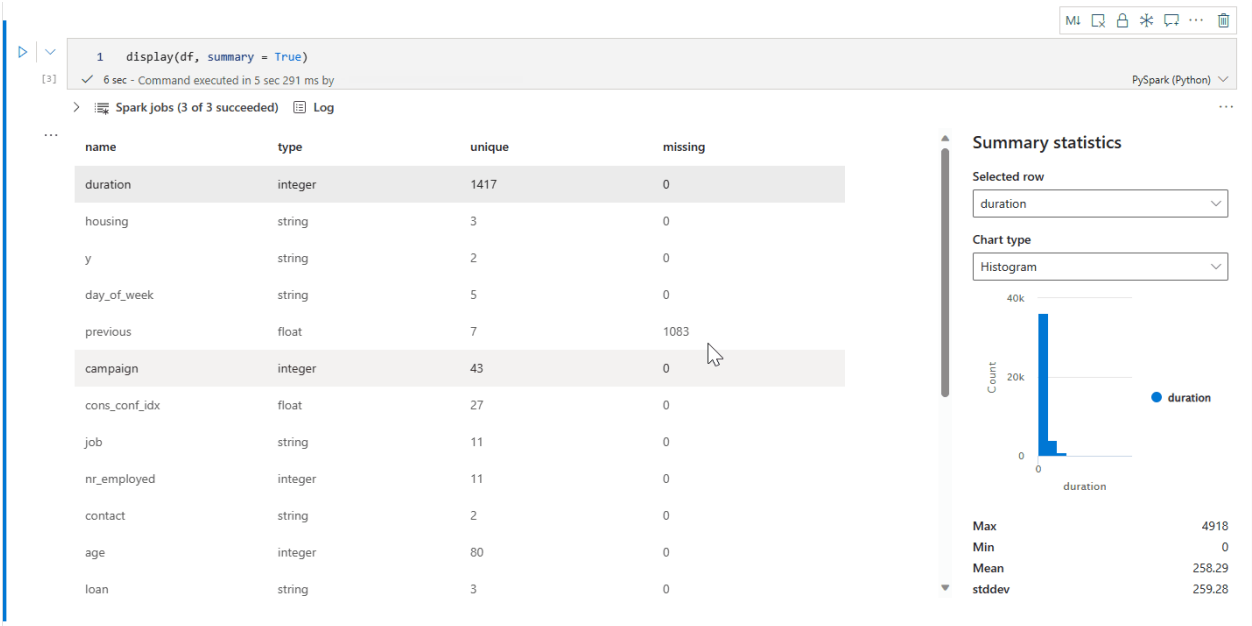
> **Note**
>
> By default the *display(df)* function will only take the first 1000 rows of the data to render the charts. Select **Aggregation over all results** and then select **Apply** to apply the chart generation from the whole dataset. A Spark job will be triggered when the chart setting changes. Please note that it may take several minutes to complete the calculation and render the chart.

4. Once done, you can view and interact with your final visualization!

display(df) summary view

You can use *display(df, summary = true)* to check the statistics summary of a given Apache Spark DataFrame that includes the column name, column type, unique values,

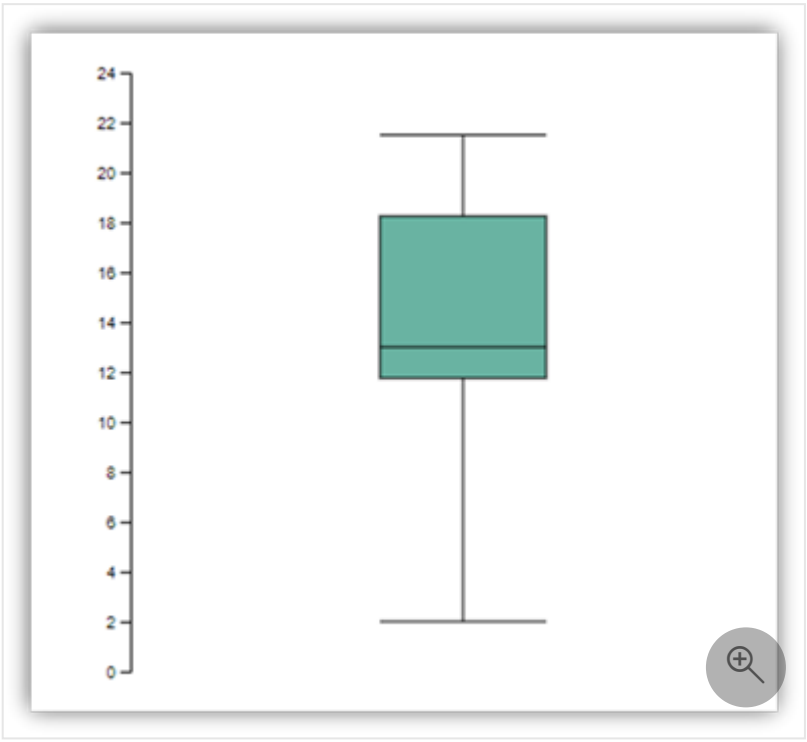
and missing values for each column. You can also select a specific column to see its minimum value, maximum value, mean value, and standard deviation.



displayHTML() option

Microsoft Fabric notebooks support HTML graphics using the *displayHTML* function.

The following image is an example of creating visualizations using D3.js .\



Run the following code to create this visualization.

Python

```
displayHTML("""<!DOCTYPE html>
<meta charset="utf-8">

<!-- Load d3.js -->
<script src="https://d3js.org/d3.v4.js"></script>

<!-- Create a div where the graph will take place -->
<div id="my_dataviz"></div>
<script>

// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 40},
  width = 400 - margin.left - margin.right,
  height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#my_dataviz")
.append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
.append("g")
  .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");

// Create Data
var data = [12,19,11,13,12,22,13,4,15,16,18,19,20,12,11,9]

// Compute summary statistics used for the box:
var data_sorted = data.sort(d3.ascending)
var q1 = d3.quantile(data_sorted, .25)
var median = d3.quantile(data_sorted, .5)
var q3 = d3.quantile(data_sorted, .75)
var interQuantileRange = q3 - q1
var min = q1 - 1.5 * interQuantileRange
var max = q1 + 1.5 * interQuantileRange

// Show the Y scale
var y = d3.scaleLinear()
  .domain([0,24])
  .range([height, 0]);
svg.call(d3.axisLeft(y))

// a few features for the box
var center = 200
var width = 100

// Show the main vertical line
svg
.append("line")
  .attr("x1", center)
  .attr("x2", center)
  .attr("y1", y(min) )
  .attr("y2", y(max) )
  .attr("stroke", "black")
```

```
// Show the box
svg
.append("rect")
  .attr("x", center - width/2)
  .attr("y", y(q3) )
  .attr("height", (y(q1)-y(q3)) )
  .attr("width", width )
  .attr("stroke", "black")
  .style("fill", "#69b3a2")

// show median, min and max horizontal lines
svg
.selectAll("toto")
.data([min, median, max])
.enter()
.append("line")
  .attr("x1", center-width/2)
  .attr("x2", center+width/2)
  .attr("y1", function(d){ return(y(d))} )
  .attr("y2", function(d){ return(y(d))} )
  .attr("stroke", "black")
</script>

"""
)
```

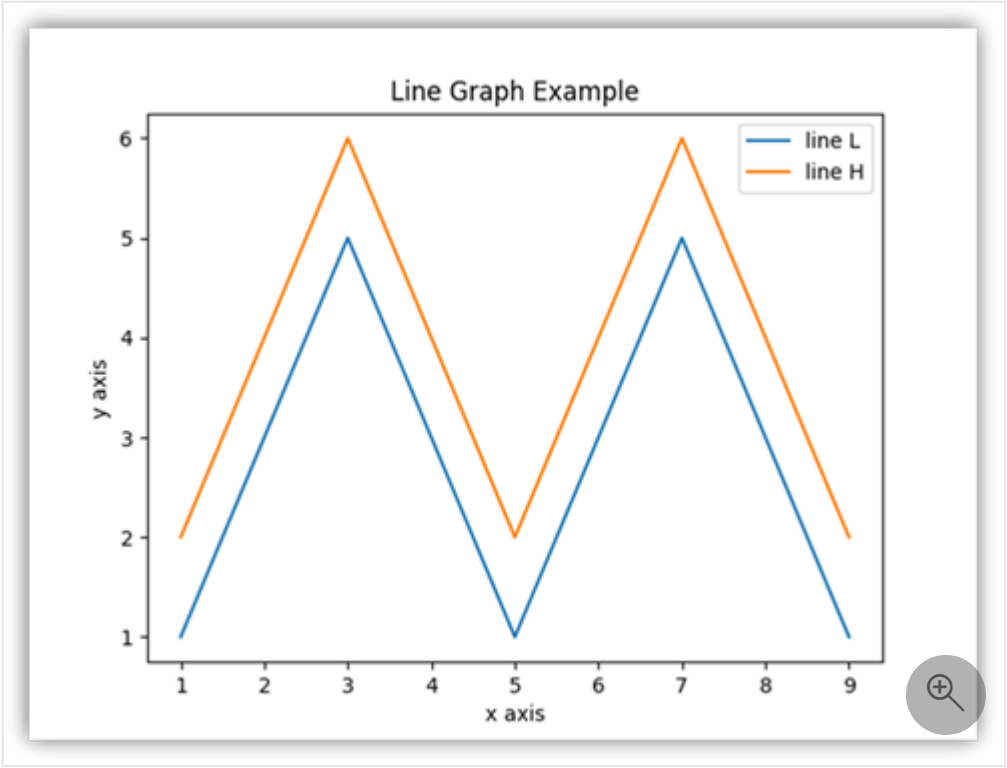
Popular libraries

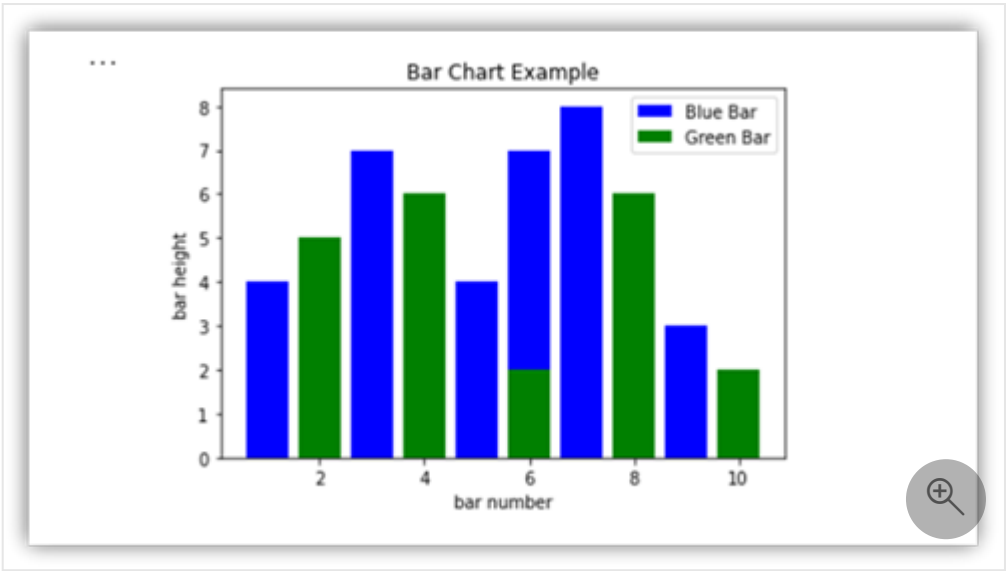
When it comes to data visualization, Python offers multiple graphing libraries that come packed with many different features. By default, every Apache Spark pool in Microsoft Fabric contains a set of curated and popular open-source libraries.

Matplotlib

You can render standard plotting libraries, like Matplotlib, using the built-in rendering functions for each library.

The following image is an example of creating a bar chart using **Matplotlib**.





Run the following sample code to draw this bar chart.

Python

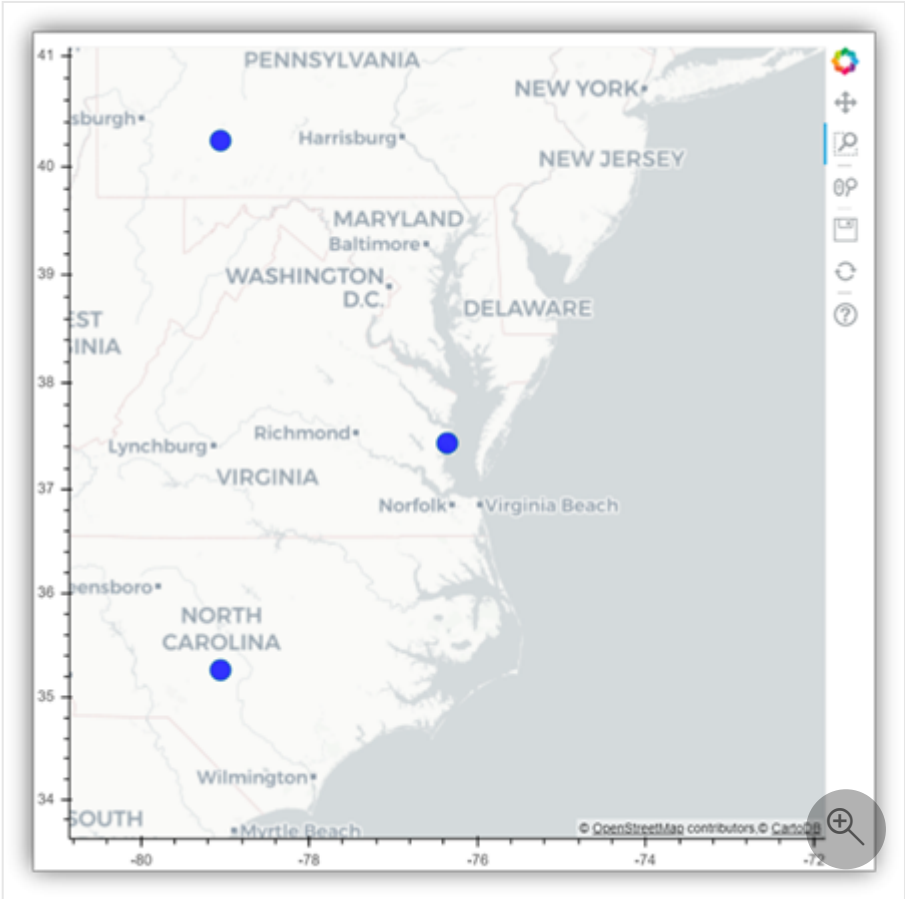
```
# Bar chart

import matplotlib.pyplot as plt

x1 = [1, 3, 4, 5, 6, 7, 9]
y1 = [4, 7, 2, 4, 7, 8, 3]

x2 = [2, 4, 6, 8, 10]
y2 = [5, 6, 2, 6, 2]

plt.bar(x1, y1, label="Blue Bar", color='b')
plt.bar(x2, y2, label="Green Bar", color='g')
```

```
plt.plot()

plt.xlabel("bar number")
plt.ylabel("bar height")
plt.title("Bar Chart Example")
plt.legend()
plt.show()
```

Bokeh

You can render HTML or interactive libraries, like **bokeh**, using the *displayHTML(df)*.

The following image is an example of plotting glyphs over a map using **bokeh**.



Run the following sample code to draw this image.

Python

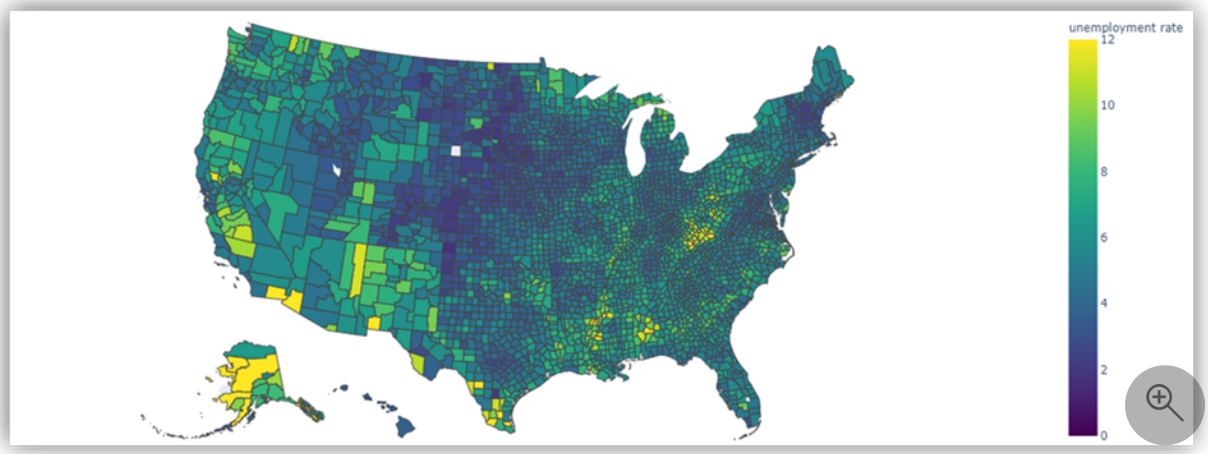
```
from bokeh.plotting import figure, output_file
from bokeh.tile_providers import get_provider, Vendors
from bokeh.embed import file_html
from bokeh.resources import CDN
from bokeh.models import ColumnDataSource

tile_provider = get_provider(Vendors.CARTODBPOSITRON)

# range bounds supplied in web mercator coordinates
```

```
p = figure(x_range=(-9000000,-8000000), y_range=(4000000,5000000),
           x_axis_type="mercator", y_axis_type="mercator")
p.add_tile(tile_provider)

# plot datapoints on the map
source = ColumnDataSource(
    data=dict(x=[ -8800000, -8500000 , -8800000],
              y=[4200000, 4500000, 4900000])
)

p.circle(x="x", y="y", size=15, fill_color="blue", fill_alpha=0.8,
source=source)

# create an html document that embeds the Bokeh plot
html = file_html(p, CDN, "my plot1")

# display this html
displayHTML(html)
```

Plotly

You can render HTML or interactive libraries like **Plotly**, using the **displayHTML()**.

Run the following sample code to draw this image:



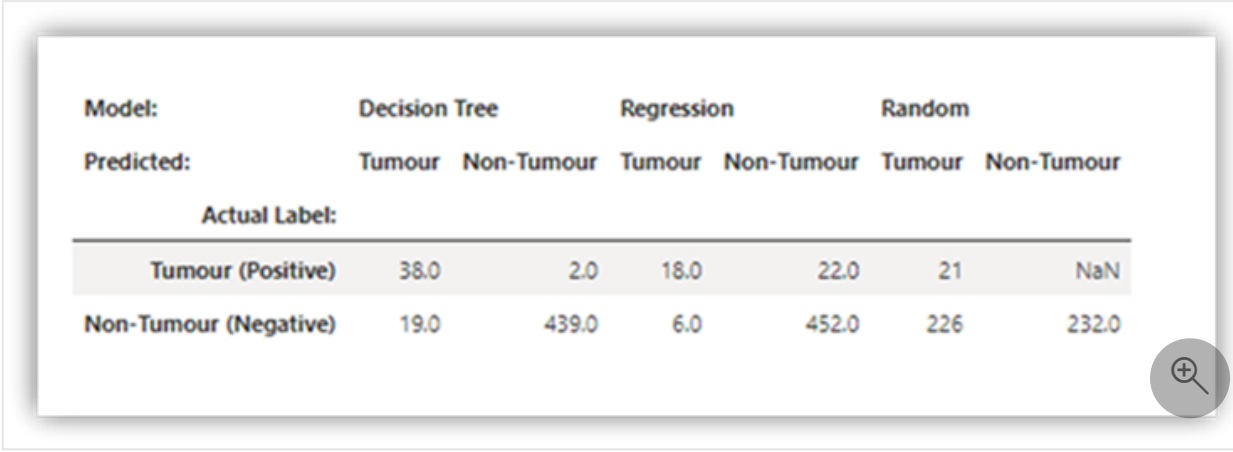
Python

```
from urllib.request import urlopen
import json
with
urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-
counties-fips.json') as response:
    counties = json.load(response)

import pandas as pd
df =
pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-
```

```
unemp-16.csv",
                    dtype={"fips": str})

import plotly
import plotly.express as px

fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
                            color_continuous_scale="Viridis",
                            range_color=(0, 12),
                            scope="usa",
                            labels={'unemp':'unemployment rate'}
                           )
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

# create an html document that embeds the Plotly plot
h = plotly.offline.plot(fig, output_type='div')

# display this html
displayHTML(h)
```

Pandas

You can view html output of pandas dataframe as the default output, notebook automatically shows the styled html content.



Model:	Decision Tree		Regression		Random	
Predicted:	Tumour	Non-Tumour	Tumour	Non-Tumour	Tumour	Non-Tumour
Actual Label:						
Tumour (Positive)	38.0	2.0	18.0	22.0	21	NaN
Non-Tumour (Negative)	19.0	439.0	6.0	452.0	226	232.0

Python

```
import pandas as pd
import numpy as np

df = pd.DataFrame([[38.0, 2.0, 18.0, 22.0, 21, np.nan],[19, 439, 6, 452,
226,232]],

                  index=pd.Index(['Tumour (Positive)', 'Non-Tumour
(Negative)'], name='Actual Label:'),

                  columns=pd.MultiIndex.from_product([['Decision Tree',
'Regression', 'Random'],['Tumour', 'Non-Tumour']], names=['Model:',
```

```
'Predicted:']))

df
```

Next steps

- Use a notebook with lakehouse to explore your data

Explore the data in your lakehouse with a notebook

Article • 05/23/2023

In this tutorial, learn how to explore the data in your lakehouse with a notebook.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Prerequisites

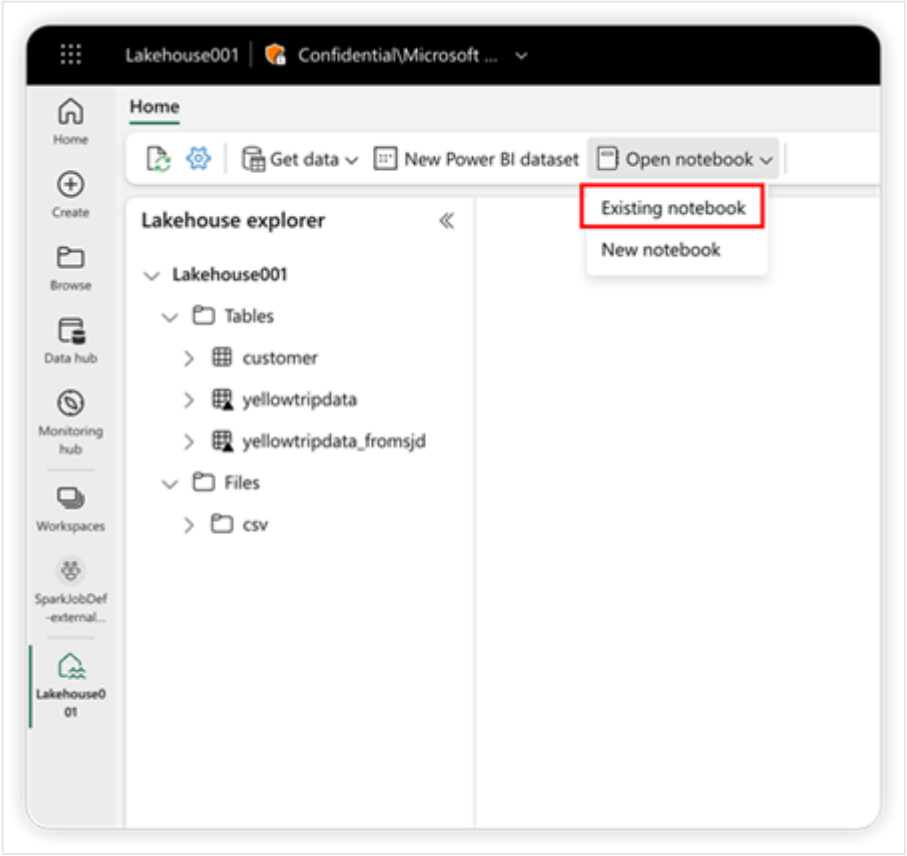
To get started, you need the following prerequisites:

- A Microsoft Fabric tenant account with an active subscription. Create an account for free.
- Read the Lakehouse overview.

Open or create a notebook from a lakehouse

To explore your lakehouse data, you can add the lakehouse to an existing notebook or create a new notebook from the lakehouse.

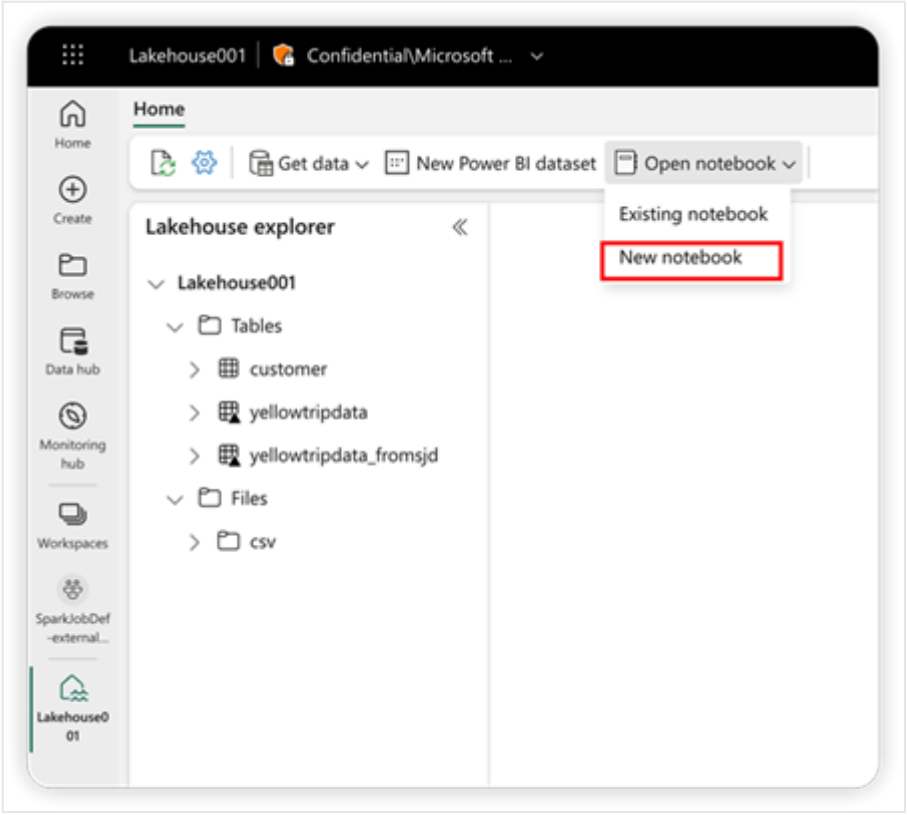
Open a lakehouse from an existing notebook



Select the notebook from the notebook list and then select **Add**. The notebook opens with your current lakehouse added to the notebook.

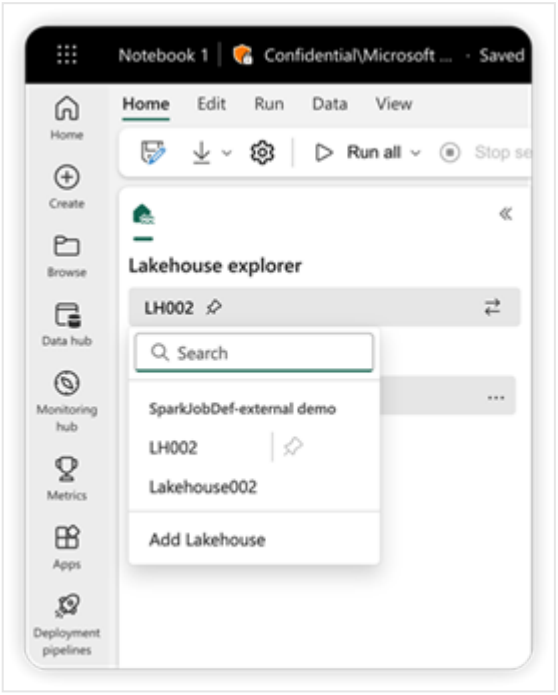
Open a lakehouse from a new notebook

You can create a new notebook in the same workspace and the current lakehouse appears in that notebook.



Switch lakehouses and set a default

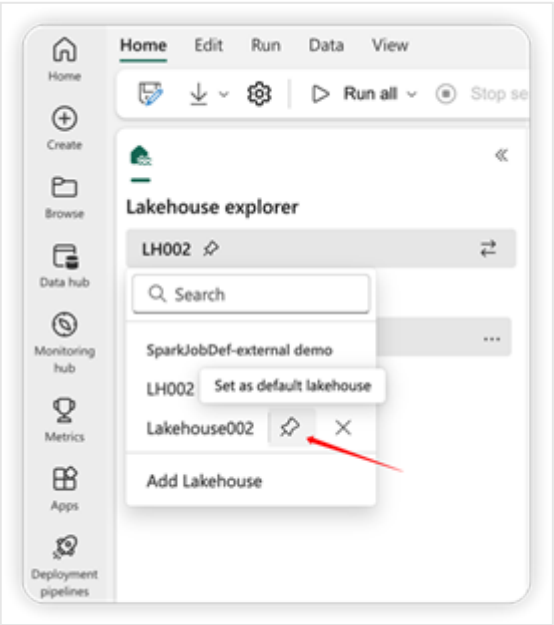
You can add multiple lakehouses to the same notebook. By switching the available lakehouse in the left panel, you can explore the structure and the data from different lakehouses.



In the lakehouse list, the pin icon next to the name of a lakehouse indicates that it's the default lakehouse in your current notebook. In the notebook code, if only a relative path

is provided to access the data from the Microsoft Fabric OneLake, then the default lakehouse is served as the root folder at run time.

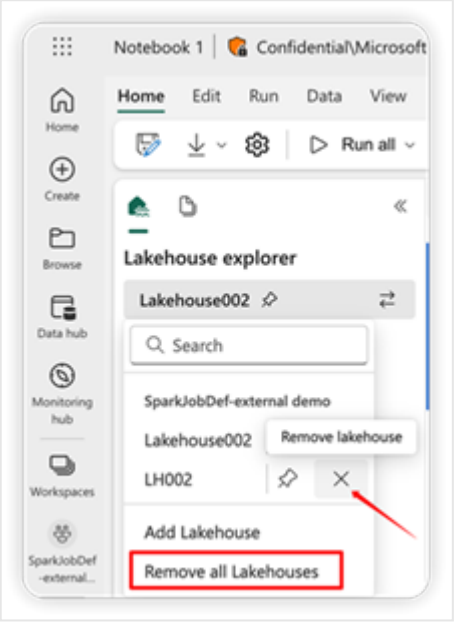
To switch to a different default lakehouse, move the pin icon.



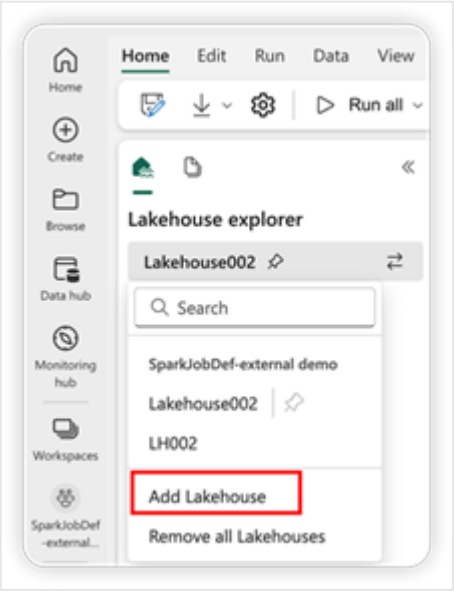
Add or remove a lakehouse

Selecting the **X** icon next to a lakehouse name removes it from the notebook, but the lakehouse item still exists in the workspace.

To remove all the lakehouses from the notebook, click "Remove all Lakehouses" in the lakehouse list.

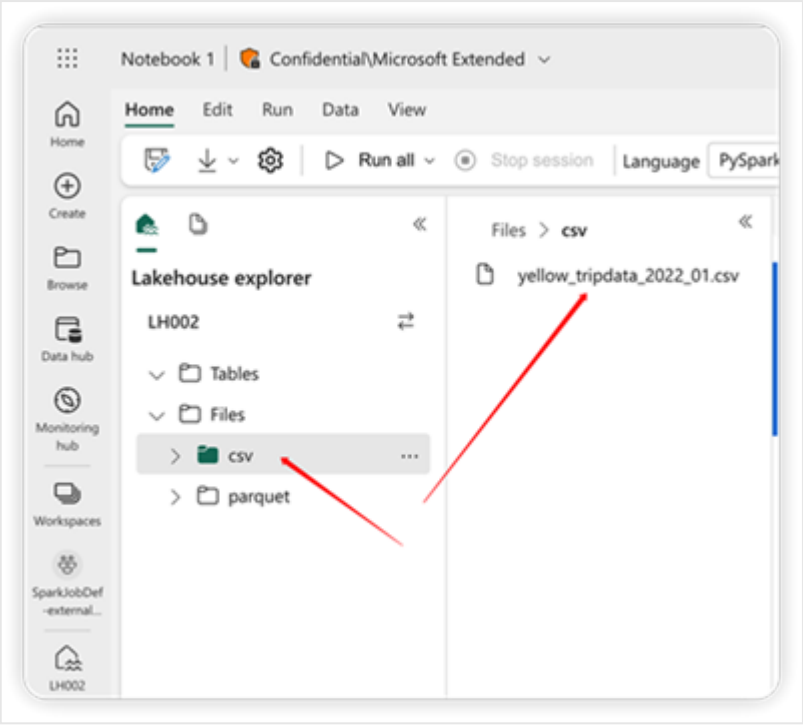


Select **Add lakehouse** to add more lakehouses to the notebook. You can either add an existing one or create a new one.



Explore the lakehouse data

The structure of the Lakehouse shown in the Notebook is the same as the one in the Lakehouse view. For the detail please check Lakehouse overview. When you select a file or folder, the content area shows the details of the selected item.



> ⓘ Note
>
> The notebook will be created under your current workspace.

Next steps

- How to use a notebook to load data into your lakehouse

Use a notebook to load data into your Lakehouse

Article • 05/23/2023

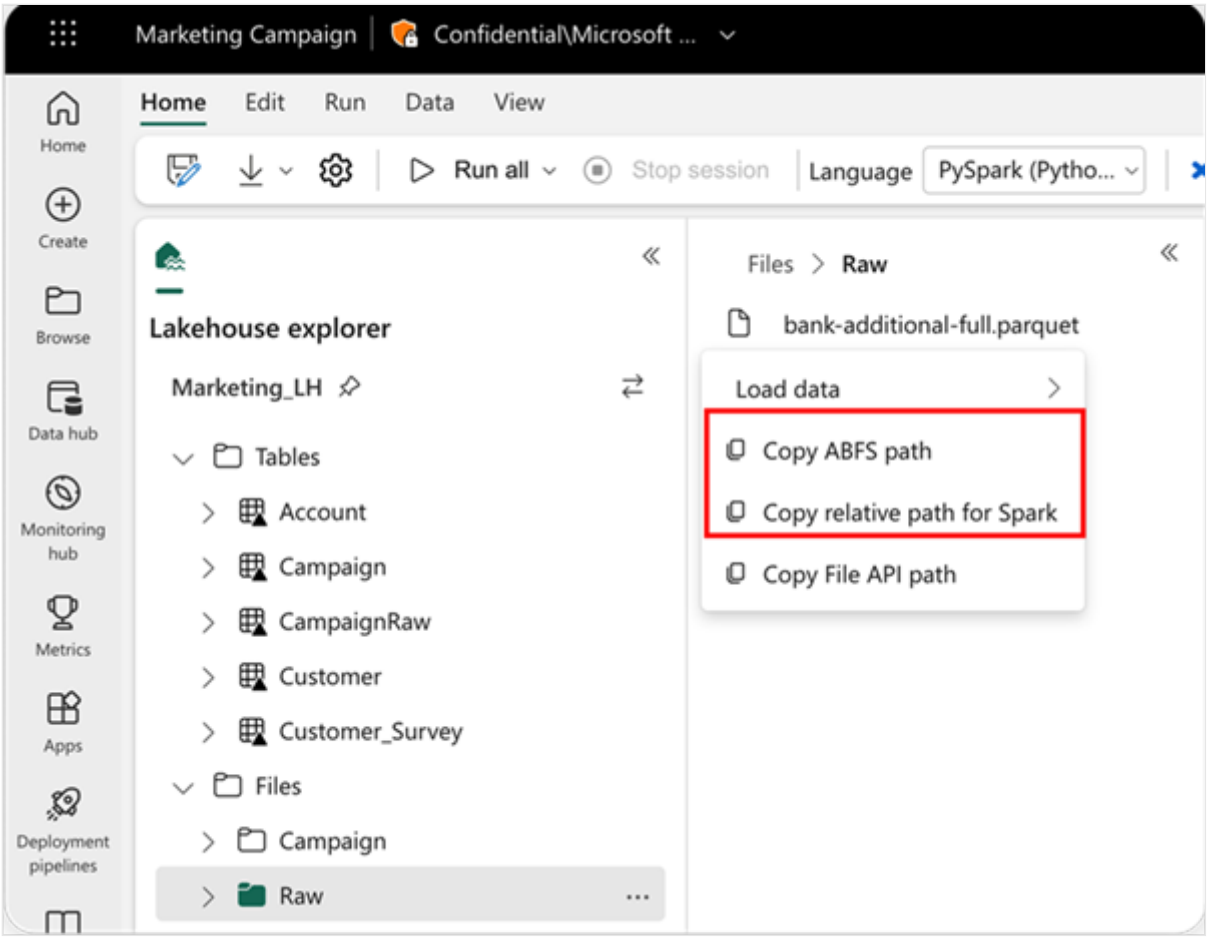
In this tutorial, learn how to read/write data into your lakehouse with a notebook.Spark API and Pandas API are supported to achieve this goal.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Load data with an Apache Spark API

In the code cell of the notebook, use the following code example to read data from the source and load it into **Files**, **Tables**, or both sections of your lakehouse.

To specify the location to read from, you can use the relative path if the data is from the default lakehouse of current notebook, or you can use the absolute ABFS path if the data is from other lakehouse. you can copy this path from the context menu of the data



Copy ABFS path : this return the absolute path of the file

Copy relative path for Spark : this return the relative path of the file in the default lakehouse

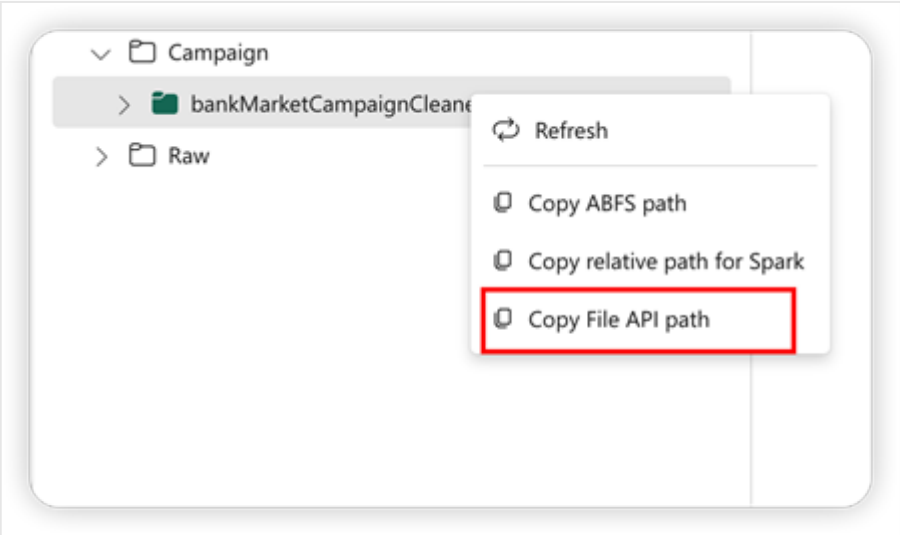
Python

```
df = spark.read.parquet("location to read from")

# Keep it if you want to save dataframe as CSV files to Files section of the default Lakehouse

df.write.mode("overwrite").format("csv").save("Files/ " + csv_table_name)

# Keep it if you want to save dataframe as Parquet files to Files section of the default Lakehouse

df.write.mode("overwrite").format("parquet").save("Files/" + parquet_table_name)

# Keep it if you want to save dataframe as a delta lake, parquet table to Tables section of the default Lakehouse

df.write.mode("overwrite").format("delta").saveAsTable(delta_table_name)

# Keep it if you want to save the dataframe as a delta lake, appending the data to an existing table
```

```
df.write.mode("append").format("delta").saveAsTable(delta_table_name)
```

Load data with a Pandas API

To support Pandas API, the default Lakehouse will be automatically mounted to the notebook. The mount point is '/lakehouse/default/'. You can use this mount point to read/write data from/to the default lakehouse. The "Copy File API Path" option from the context menu will return the File API path from that mount point. The path returned from the option **Copy ABFS path** also works for Pandas API.



Copy File API Path :This return the path under the mount point of the default lakehouse

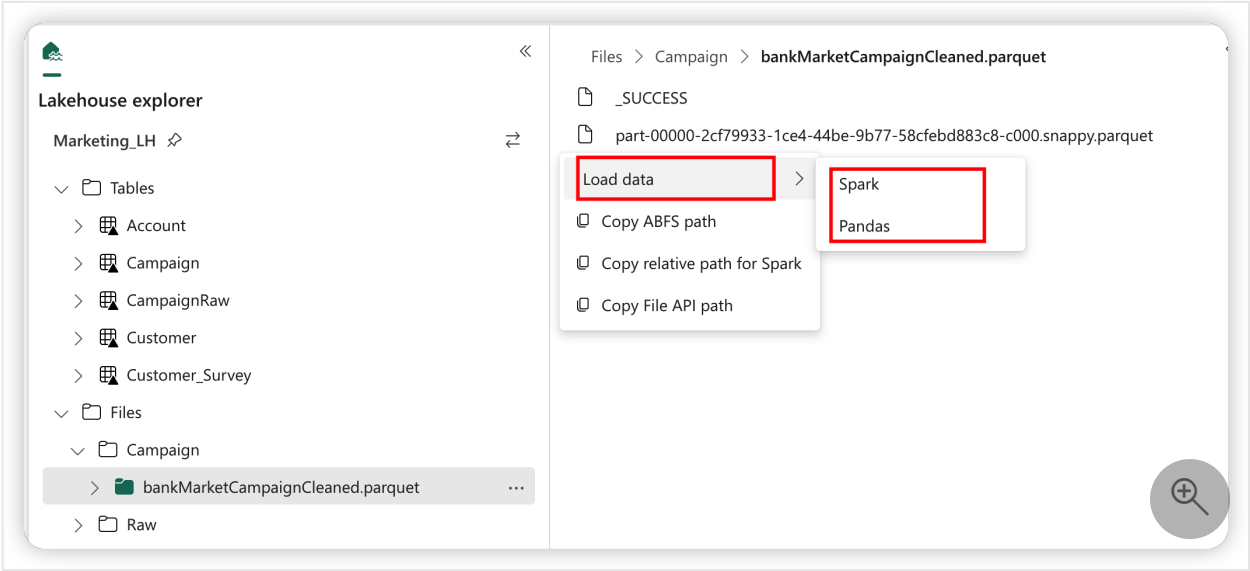
Python

```
# Keep it if you want to read parquet file with Pandas from the default
lakehouse mount point

import pandas as pd
df = pd.read_parquet("/lakehouse/default/Files/sample.parquet")

# Keep it if you want to read parquet file with Pandas from the absolute
abfss path

import pandas as pd
df = pd.read_parquet("abfss://DevExpBuildDemo@msit-
onelake.dfs.fabric.microsoft.com/Marketing_LH.Lakehouse/Files/sample.parquet
")
```

Tip

For Spark API, please use the option of **Copy ABFS path** or **Copy relative path for Spark** to get the path of the file. For Pandas API, please use the option of **Copy ABFS path** or **Copy File API path** to get the path of the file.

The quickest way to have the code to work with Spark API or Pandas API is to use the option of **Load data** and select the API you want to use. The code will be automatically generated in a new code cell of the notebook.



Next steps

- Explore the data in your lakehouse with a notebook

What is the Synapse Visual Studio Code extension?

Article • 05/23/2023

The Synapse VS Code extension supports a pro-developer experience for exploring Microsoft Fabric lakehouses, and authoring Fabric notebooks and Spark job definitions. Learn more about the extension, including how to get started with the necessary prerequisites.

> **Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Visual Studio Code is a one of the most popular lightweight source code editors; it runs on your desktop and is available for Windows, macOS, and Linux. By installing the Synapse VS Code extension, you can author, run, and debug your notebook and Spark job definition locally in VS Code. You can also post the code to the remote Spark compute in your Fabric workspace to run or debug. The extension also allows you to browse your lakehouse data, including tables and raw files, in VS Code.

Prerequisites

Prerequisites for the Synapse VS Code extension:

- Java 1.8
- Conda
- Jupyter extension for VS Code

After you have installed the required software, you must update the operating system properties.

Windows

1. Add **JAVA_HOME** to the environment variables and point it to the directory where java 1.8 is installed.

2. Add both **%JAVA_HOME%/bin** and the **Contain** subfolder of the Conda installation to the system path directory.

macOS

Run the **conda.sh** in the terminal:

1. Open the terminal window, change the directory to the folder where conda is installed, then change to the subdirectory **etc/profile.d**. The subdirectory should contain a file named conda.sh.

2. Execute `Source conda.sh`.

3. In the same terminal window, run `sudo conda init`.

4. Type in `Java --version`. The version should be Java 1.8.

Install the extension and prepare your environment

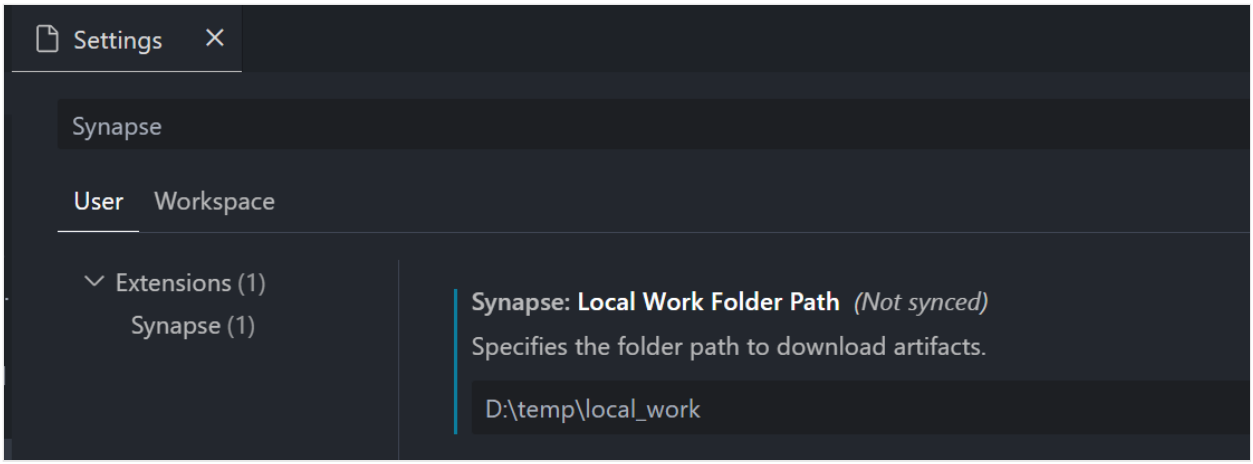
1. Search for **Synapse VS Code** in the VS Code extension marketplace and install the extension. (The extension is still under preview, so you need to select the prerelease version to install.)

2. After the extension installation is complete, restart VS Code. The icon for the extension is listed at the VS Code activity bar.

Local working directory

To edit a notebook, you must have a local copy of the notebook content. The local working directory of the extension serves as the local root folder for all downloaded notebooks, even notebooks from different workspaces. By invoking the command `Synapse:Set Local Work Folder`, you can specify a folder as the local working directory for the extension.

To validate the setup, open the extension settings and check the details there:

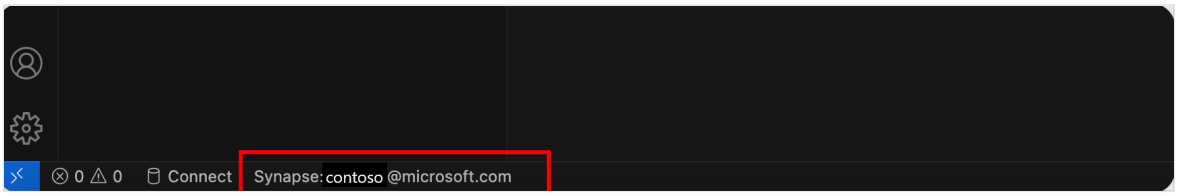


Sign in and out of your account

1. From the VS Code command palette, enter the `Synapse:Sign in` command to sign in to the extension. A separate browser sign-in page appears.

2. Enter your username and password.

3. After you successfully sign in, your username will be displayed in the VS Code status bar to indicate that you're signed in.

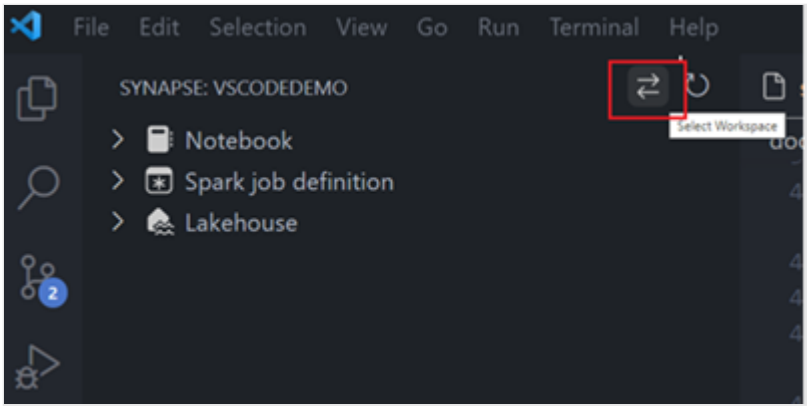
   

4. To sign out of the extension, enter the command `Synapse: Sign off`.

Choose a workspace to work with

To select a Fabric workspace, you must have a workspace created. If you don't have one, you can create one in the Fabric portal. For more information, see Create a workspace.

Once you have a workspace, choose it by selecting the **Select Workspace** option. A list appears of all workspaces that you have access to; select the one you want from the list.



Next steps

In this overview, you get a basic understanding of how to install and set up the Synapse VS Code extension. The next articles explain how to develop your notebooks and Spark job definitions locally in VS Code.

- To get started with notebooks, see Microsoft Fabric notebook experience in VS Code.
- To get started with Spark job definitions, see Spark job definition experience in VS Code.

Microsoft Fabric notebook experience in VS Code

Article • 05/23/2023

The Visual Studio Code extension for Synapse fully supports the CURD (create, update, read, and delete) notebook experience in Fabric. The extension also supports synchronization between local and remote workspaces; when you synchronize changes, you can address any conflicts or differences between your local and remote workspace.

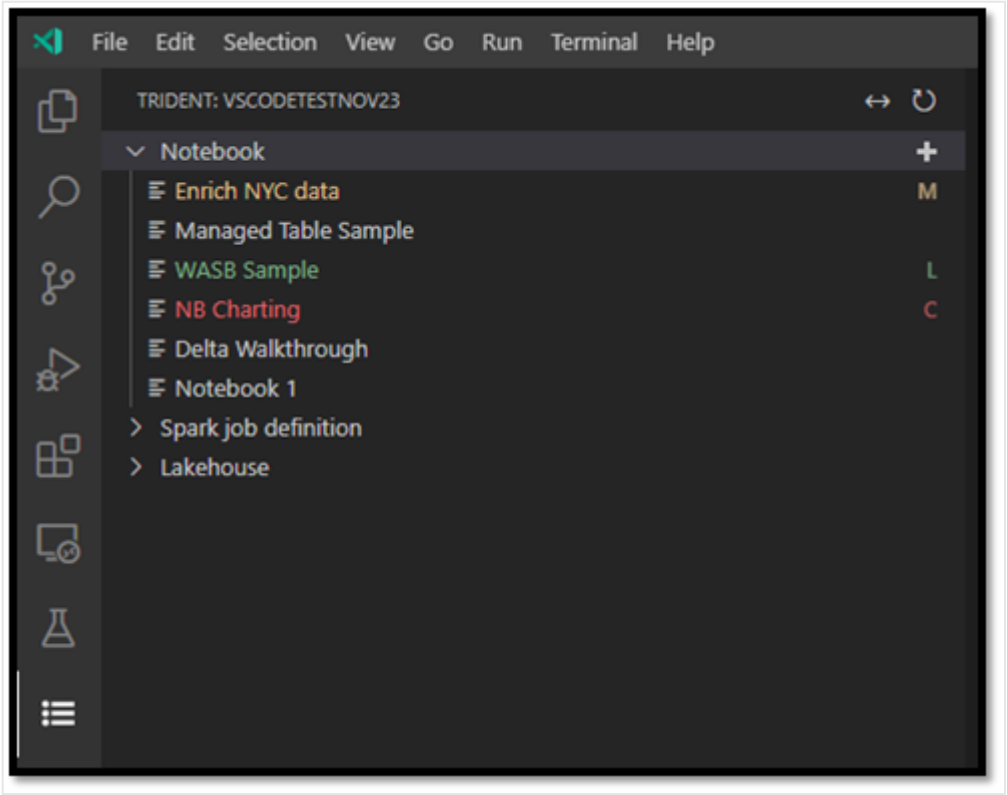
With this extension, you can also run notebooks onto the remote Fabric Spark compute.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

View the list of notebooks

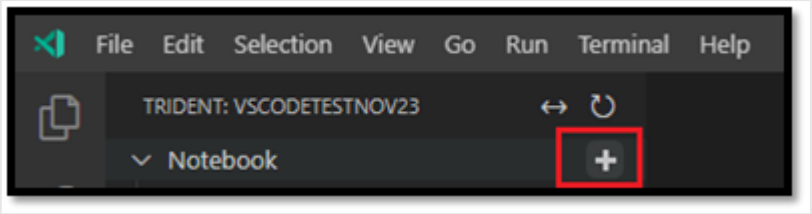
The notebook tree node lists the names of the all the notebook items in the current workspace. Based on your changes in VS Code, the list displays different colors and characters to indicate the latest state, as shown in the following image.

- Default: White text and no character to the right of the notebook name indicates the default or initialized state. The notebook exists in the remote workspace and hasn't been downloaded locally.
- Modified: The **M** character to the right of the name and yellow text indicates the notebook has been downloaded and edited locally in VS Code, and those pending changes have yet to be published back to the remote workspace.
- Local: The **L** character and green text indicates the notebook has been downloaded and the content is the same as the remote workspace.
- Conflict: The **C** character and red text indicates that conflicts exist between the local version and the remote workspace version.



Create a notebook

1. In VS Code Explorer, hover over the notebook toolbar. The **Create Notebook** option appears.

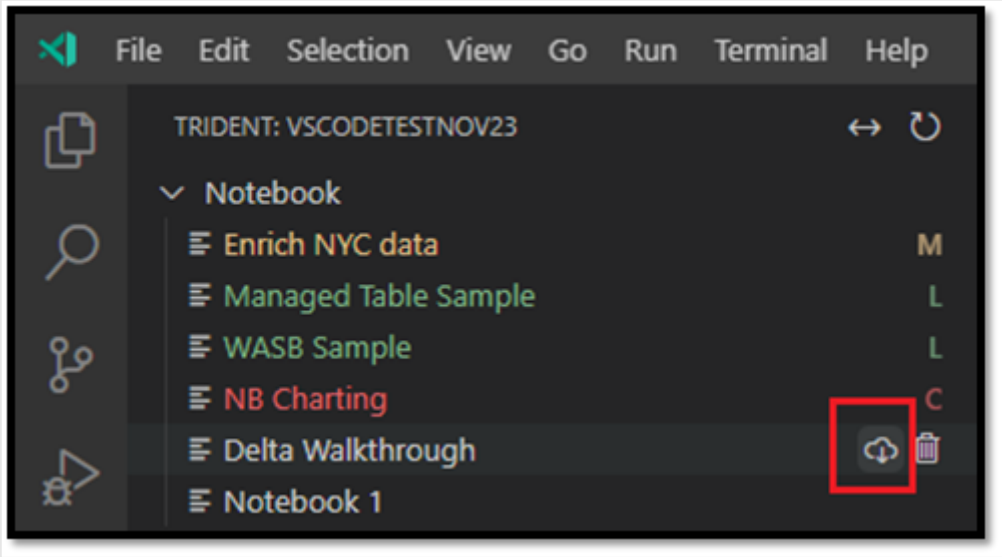


2. Select **Create Notebook** and enter a name and description. A new notebook is created in the remote workspace and appears in your notebook list in the default state.

Download a notebook

Before you can edit the notebook content, you must download the notebook to VS Code.

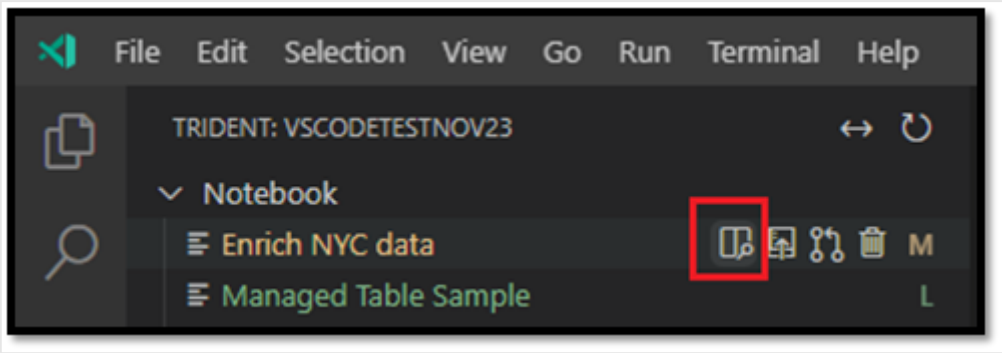
1. In the notebook list in VS Code, hover over the notebook name. The **Download** option appears beside the notebook name.



2. Select **Download** and save the notebook to your local working directory.

Open a notebook

1. In VS Code Explorer, hover over the name of a downloaded notebook. Several options appear next to the notebook, including the **Open Notebook Folder** option.



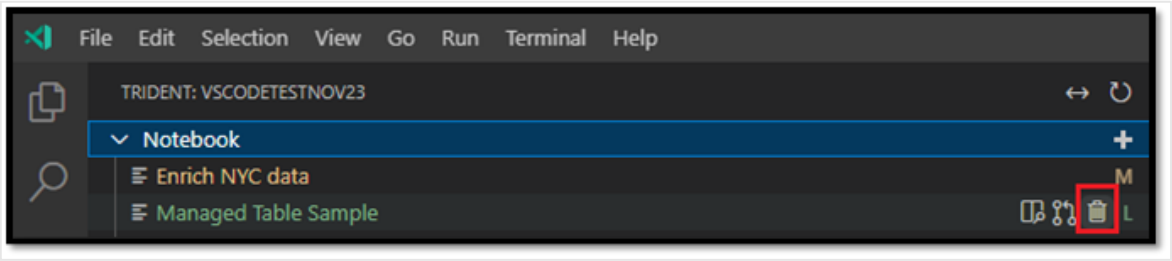
2. Select **Open Notebook Folder** and the notebook opens in the VS Code Editor screen.

Delete a notebook

> **Tip**
>
> To avoid failure, close the notebook folder in the Explorer view and close the notebook in the editor view before deleting the notebook.

To delete a notebook:

1. In VS Code Explorer, hover over the name of the notebook you want to delete; options appear to the right of the name, including the **Delete Notebook** option.

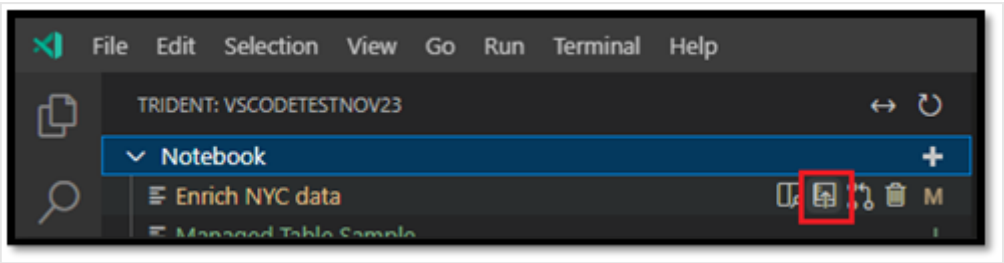


2. Select the **Delete Notebook** option. When prompted, choose to delete only the local copy or both the local and the remote workspace copies.

Publish local changes to the remote workspace

To push your local changes to the remote workspace:

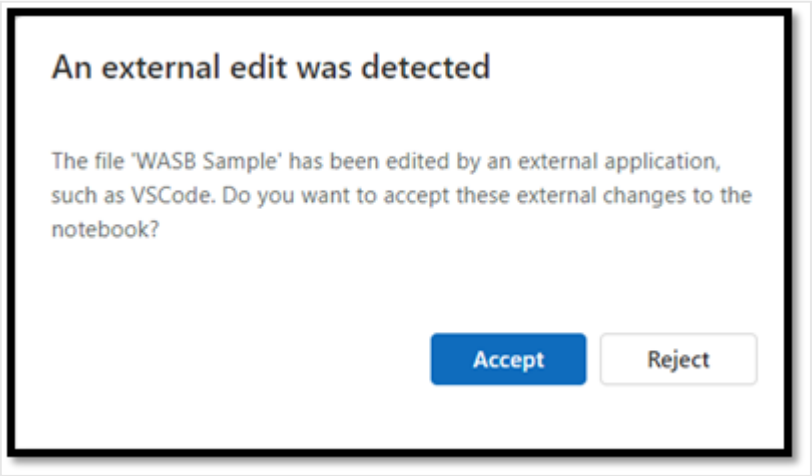
1. In VS Code Explorer, hover over the name of the notebook you want to publish to the remote workspace; options appear to the right of the name, including the **Publish** option.



2. Select **Publish**. The remote workspace version is updated with your local VS Code changes.

    - If your local update creates any merge conflicts, you're prompted to resolve them before the merge goes through.

3. If someone else has the same notebook open in the Fabric portal, they're notified to accept or reject your local VS Code changes, as shown in the following image.

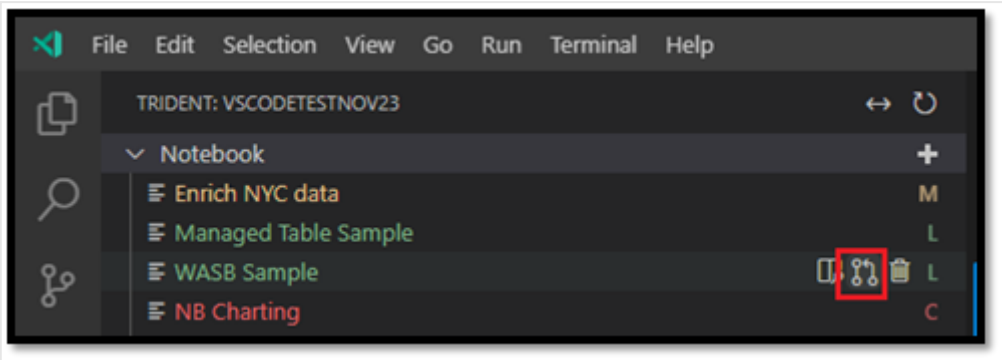


- **Accept**: your change from VS Code is successfully saved in the workspace.
- **Reject**: your change from VS Code is ignored.

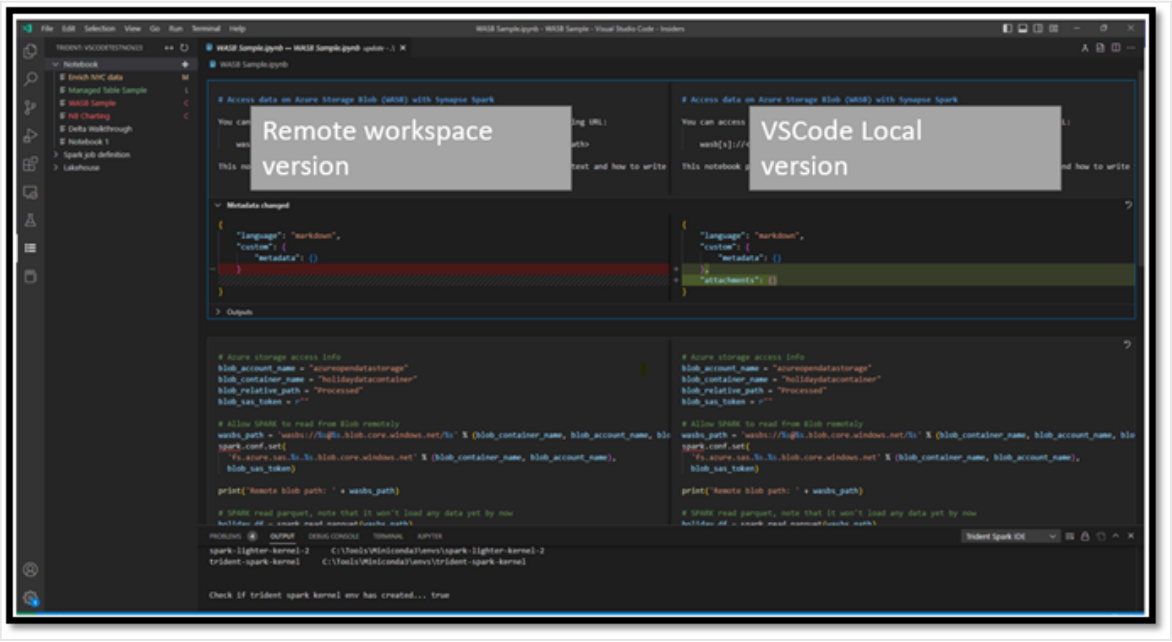
Pull changes from the remote workspace

To update your local version with the latest workspace version, you pull the remote version:

1. In VS Code Explorer, hover over the name of the notebook you want to update; options appear to the right of the name, including the **Update Notebook** option.

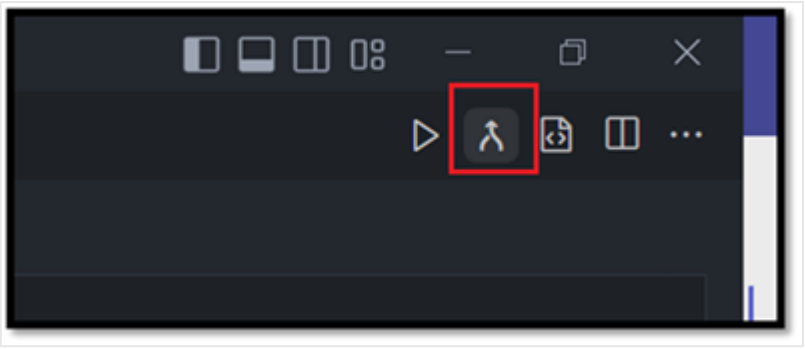


2. VS Code pulls the latest version from the remote workspace, and opens the VS Code diff editor to compare the two notebook files. The left side screen is from the workspace, the right side screen is from the local version:



3. Update the code/markdown cell on the left side to address the issue.

4. When all conflicts are addressed, select the **Merge** option at the top-right corner of the diff editor to confirm the merge is complete. (Until you select **Merge**, the notebook stays in **Conflict** mode.

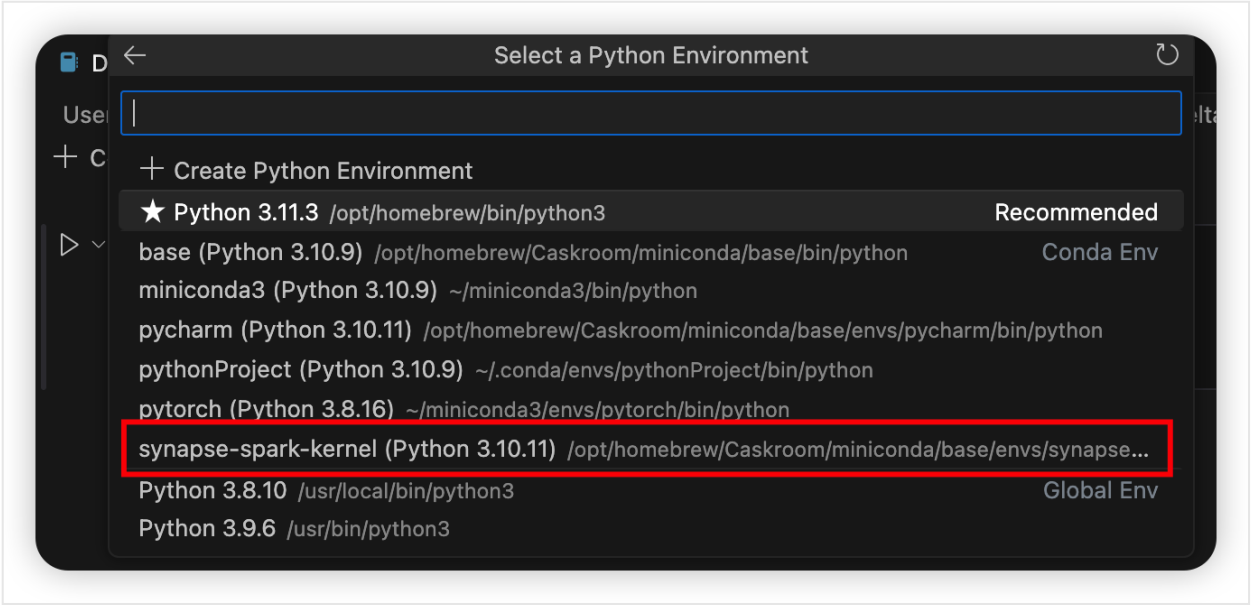


> **ⓘ Important**
>
> After the diff editor is opened once, the extension will NOT automatically refresh the left side of the diff view to fetch the latest update from the remote workspace.

Run or debug a notebook on remote Spark compute

By selecting the kernel **synapse-spark-kernel** shipped with this extension, you can run the code cell on top of the remote Fabric Spark compute. Once this kernel is selected, during runtime, the extension intercepts all the PySpark API calls and translates them to

the corresponding http call to the remote Spark compute. For pure python code, it's still executed in the local environment.



Next steps

- Spark Job Definition experience in VS Code
- Explore lakehouse from VS Code

Spark job definition experience in VS Code

Article • 05/23/2023

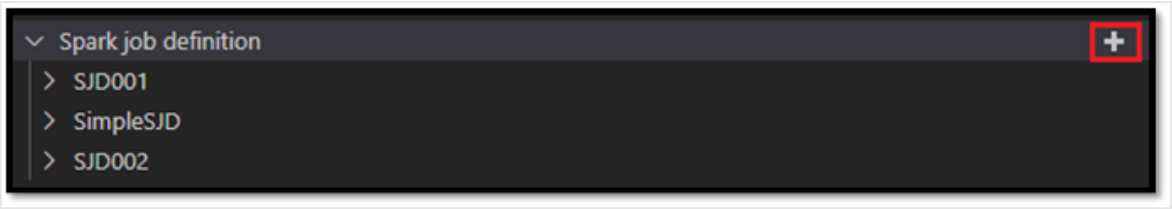
The Visual Studio Code extension for Synapse fully supports the CURD (create, update, read, and delete) Spark job definition experience in Fabric. After you create a Spark job definition, you can upload more referenced libraries, submit a request to run the Spark job definition, and check the run history.

> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Create a Spark job definition

To create a new Spark job definition:

1. In the VS Code Explorer, select the **Create Spark Job Definition** option.



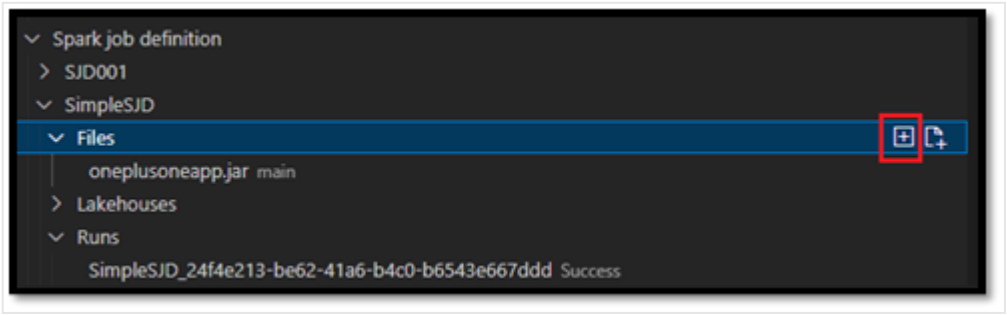
2. Enter the initial required fields: name, referenced lakehouse, and default lakehouse.

3. After the request is processed, the name of your newly created SJD appears under the **Spark Job Definition** root node in VS Code Explorer. Under the Spark job definition name node, three subnodes are listed:

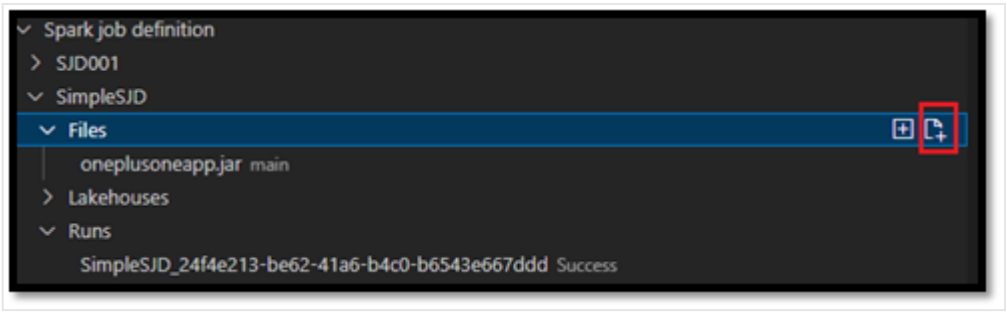
    - **Files**: List of the main definition file and other referenced libraries. You can upload new files from this list.
    - **Lakehouse**: List of all lakehouses referenced by this Spark job definition. The default lakehouse is marked in the list, and you can access it via the relative path `Files/…, Tables/…`.
    - **Run**: List of the run history of this Spark job definition and the job status of each run.

Upload a main definition file to a referenced library

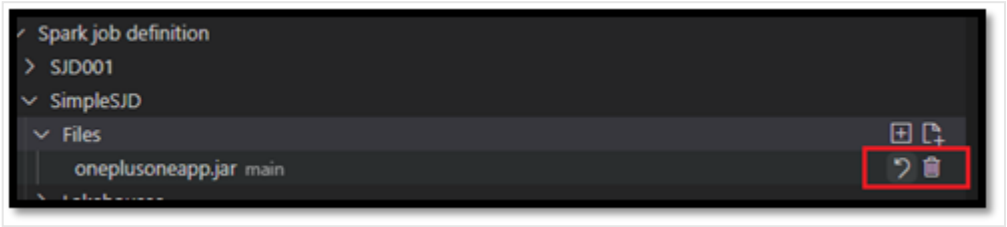
To upload or overwrite the main definition file, select the **Add Main File** option.



To upload the library file that is referenced in the main definition file, select the **Add Lib File** option.



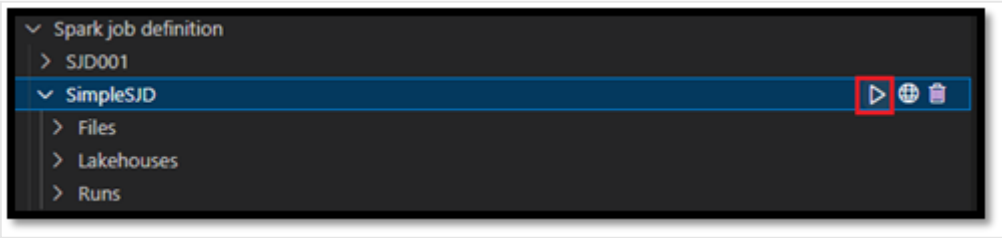
After a file is uploaded, you can override it by clicking the **Update File** option and uploading a new file, or just delete the file via the **Delete** option.



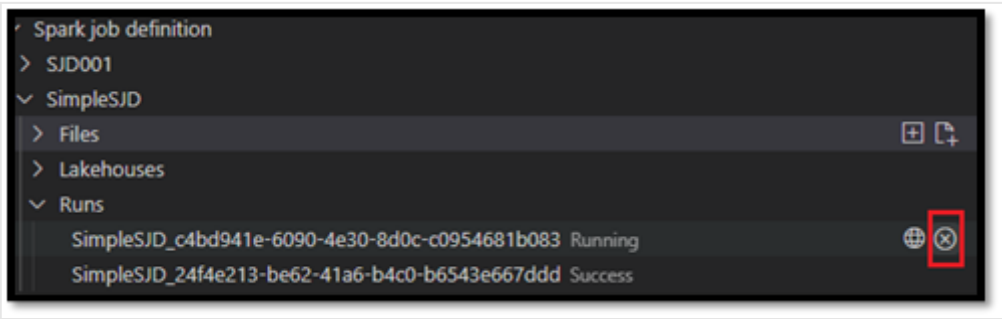
Submit a run request

To submit a request to run the Spark job definition from VS Code:

1. From the options to the right of the name of the Spark job definition you want to run, select the **Run Spark Job** option.



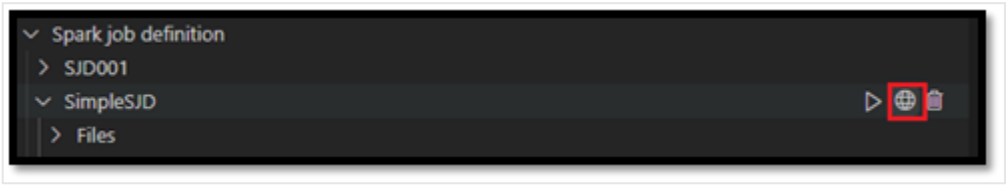
2. After you submit the request, a new Spark Application appears in the **Runs** node in the Explorer list. You can cancel the running job by selecting the **Cancel Spark Job** option.



Open a Spark job definition in the Fabric portal

You can open the Spark job definition authoring page in the Fabric portal by selecting the **Open in Browser** option.

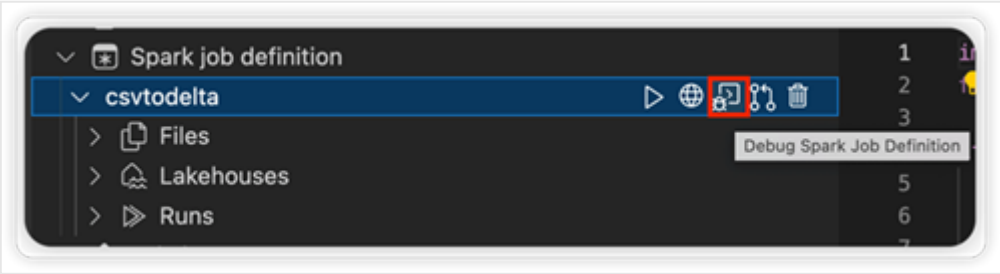
You can also select **Open in Browser** next to a completed run to see the detail monitor page of that run.



Debug Spark job definition source code (Python)

If the Spark job definition is created with PySpark (Python), you can download the .py script of the main definition file and the referenced file, and debug the source script in VS Code.

1. To download the source code, select the **Debug Spark Job Definition** option to the right of the Spark job definition.



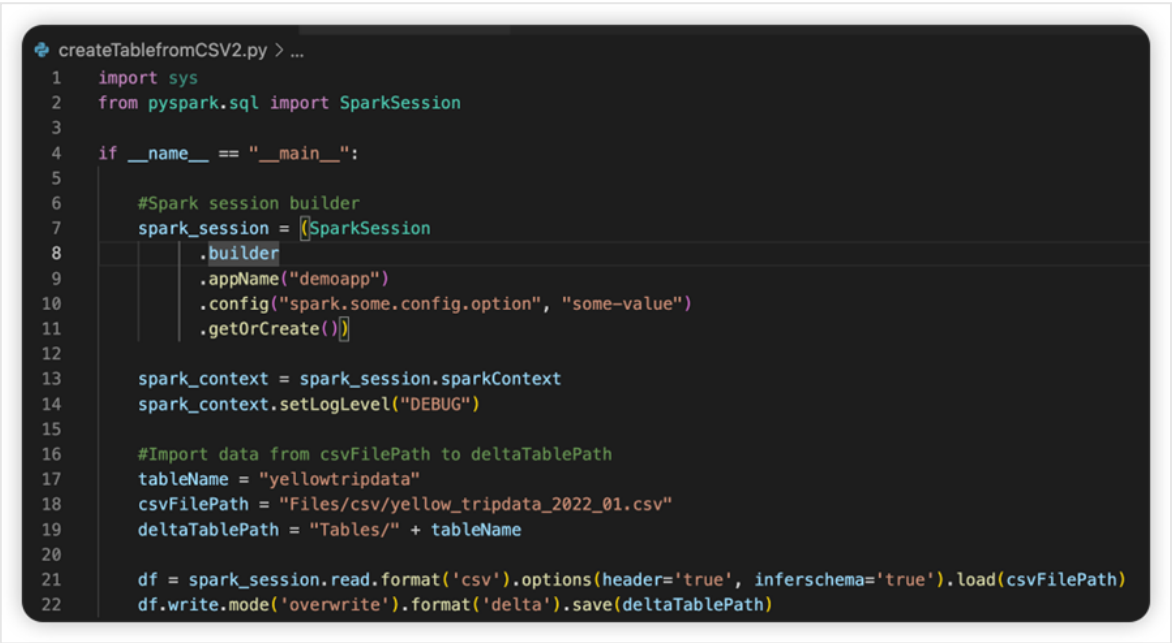
2. After the download is finished, the folder of the source code automatically opens.

3. Select the **Trust the authors** option when prompted. (This option only appears the first time you open the folder. If you don't select this option, you won't be able to debug or run the source script. For more information, see Visual Studio Code Workspace Trust security .)

4. If you have downloaded the source code before, you're prompted to confirm that the local version should be overwritten by the new download.

> **Note**
>
> In the root folder of the source script, the system creates a subfolder named **conf**. Within this folder, a file named **lighter-config.json** contains some system metadata needed for the remote run. Do NOT make any changes to it.

5. The file named **sparkconf.py** contains a code snippet that you need to add to set up the **SparkConf** object. To enable the remote debug, make sure the **SparkConf** object is set up properly. The following image shows the original version of the source code.



The next image is the updated source code after you copy and paste the snippet.

createTablefromCSV2.py > ...

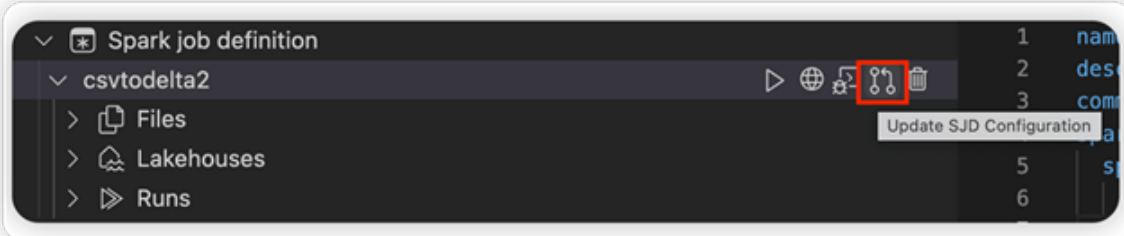
```python
import sys
from pyspark.sql import SparkSession
from pyspark.conf import SparkConf

if __name__ == "__main__":

    #Spark session builder
    conf = SparkConf()
    conf.set("spark.lighter.client.plugin", "org.apache.spark.lighter.DefaultLighterClientPlugin")
    conf.set("spark.sql.catalogImplementation", "lighter")
    conf.set("spark.lighter.sessionState.implementation", "org.apache.spark.sql.lighter.client.SparkLighterSessionStateBuilder")
    conf.set("spark.lighter.externalCatalog.implementation", "org.apache.spark.sql.lighter.client.ConnectCatalogClient")
    spark_session = (SparkSession
          .builder
          .appName("demoapp")
          .config(conf=conf)
          .getOrCreate())

    spark_context = spark_session.sparkContext
    spark_context.setLogLevel("DEBUG")

    #Import data from csvFilePath to deltaTablePath
    tableName = "yellowtripdata"
    csvFilePath = "Files/csv/yellow_tripdata_2022_01.csv"
    deltaTablePath = "Tables/" + tableName

    df = spark_session.read.format('csv').options(header='true', inferschema='true').load(csvFilePath)
    df.write.mode('overwrite').format('delta').save(deltaTablePath)
```

6. After you have updated the source code with the necessary conf, you must pick the right Python Interpreter. Make sure to select the one installed from the **synapse-spark-kernel** conda environment.

Edit Spark Job Definition properties

You can edit the detail properties of Spark job definitions, such as command-line arguments.

1. Select the **Update SJD Configuration** option to open a **settings.yml** file. The existing properties populate the contents of this file.



2. Update and save the .yml file.

3. Select the **Publish SJD Property** option at the top right corner to sync the change back to the remote workspace.

Next steps

- Explore lakehouse in VS Code
- Notebook experience in VS Code

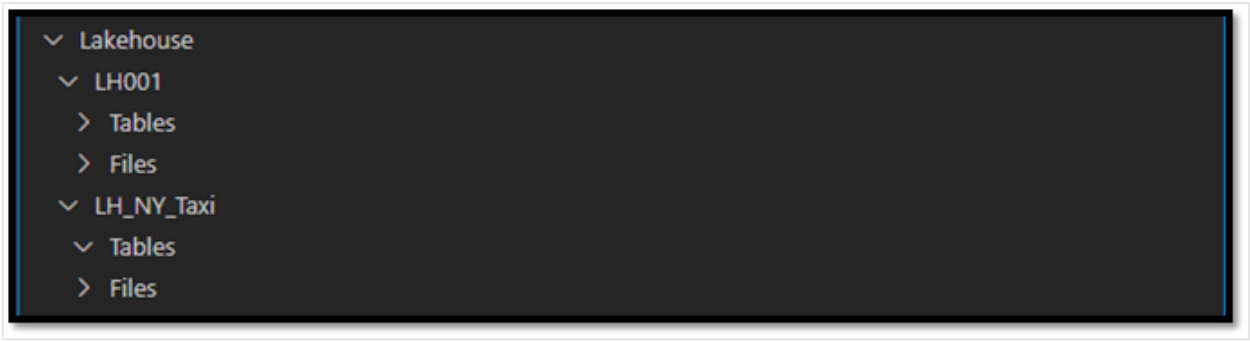
Explore Microsoft Fabric lakehouses in VS Code

Article • 05/23/2023

You can use the Synapse VS Code extension to explore the structure of your lakehouse in a workspace.

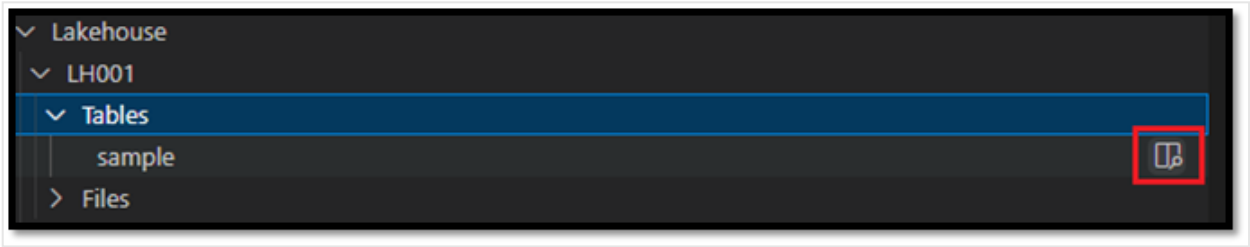
> **ⓘ Important**
>
> Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

The extension displays the lakehouse structure in a tree view that includes the **Files** and **Tables** sections. All the lakehouses from the workspace you select are listed under the lakehouse root tree node.

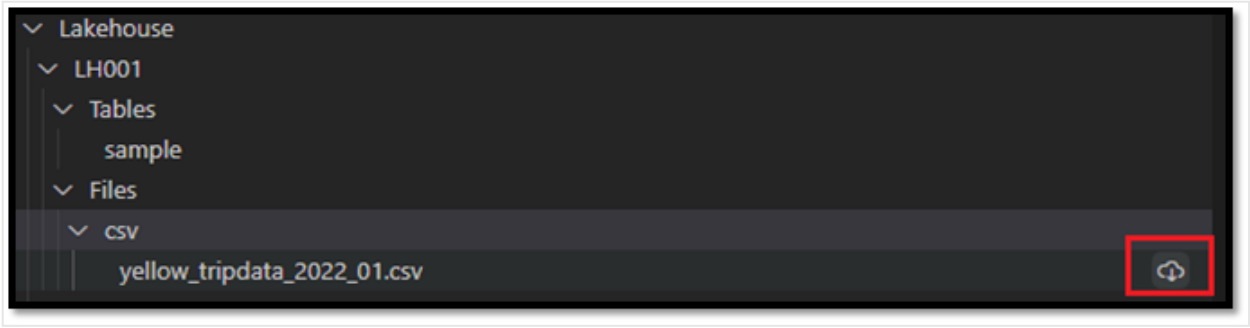


Explore a lakehouse and preview table data

Expand the **Tables** node to find the table entities from the lakehouse. To review the first 100 rows of a specific table, select the **Preview Table** option to the right of the table.

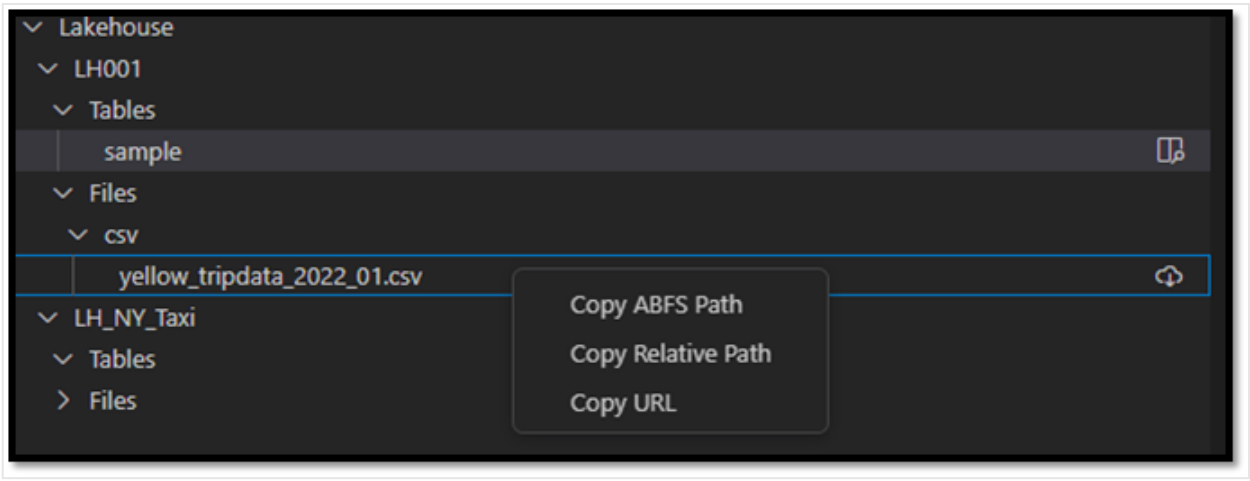


Expand the **Files** node to find the folder and files that are saved in the lakehouse. Select the **Download** option to the right of a file name to download that file.



Copy the lakehouse path

To make it easier to reference a table, folder, or file in the code, you can copy the relevant path. Right click on the target node and to find the options to **Copy ABFS path**, **Copy Relative Path**, or **Copy URL**.



Next steps

- What is SQL Endpoint for a lakehouse?
- Get data into the Fabric Lakehouse